Python绝技

运用Python成为
顶级数据工程师

黄文青◎编著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

VISIT…

LANZAROTE Caliente.com

内 容 简 介

Python 已成为广受数据科学领域欢迎的开发语言。本书契合这一趋势，结合具体的业务场景，从数据思维的角度出发，剖析各业务环节中数据处理的策略、算法，并运用 Python 代码呈现翔实的案例，构建出一个完整的数据分析体系。

在内容的组织和安排上，本书层次分明、详略得当：针对简单的数据分析工作，读者可以先浏览第 1 章至第 3 章；专职从事数据分析的工程师可以通篇阅读本书，以构建数据处理工程的完整知识框架；本书的最后一章针对从事大数据分析的工程师提供了一些常见问题的解决思路和方法。

本书既适合刚接触数据工程的从业人员作为入门参考，也可以帮助具有一定经验的数据工程师搭建知识体系，洞悉业务场景中的数据奥秘，得心应手地运用数据指导业务。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目（CIP）数据

Python 绝技：运用 Python 成为顶级数据工程师 / 黄文青编著. —北京：电子工业出版社，2018.6
ISBN 978-7-121-33654-6

Ⅰ. ①P… Ⅱ. ①黄… Ⅲ. ①软件工具－程序设计 Ⅳ. ①TP311.561

中国版本图书馆 CIP 数据核字(2018)第 024601 号

责任编辑：刘　皎
印　　刷：三河市华成印务有限公司
装　　订：三河市华成印务有限公司
出版发行：电子工业出版社
　　　　　北京市海淀区万寿路 173 信箱　邮编 100036
开　　本：720×1000　1/16　印张：13.5　字数：232 千字
版　　次：2018 年 6 月第 1 版
印　　次：2018 年 6 月第 1 次印刷
定　　价：79.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819，faq@phei.com.cn。

好评袭来

数据工程师必备三大技能：数据工程能力、数据分析能力、业务能力，三者相辅相承，缺一不可。本书从这三个方面出发，以简单易懂的 Python 为基础工具，介绍了很多基础知识和工程案例，读起来非常痛快！

——路人甲，公众号“一个程序员的日常”

基于开源的第三方库和活跃的社区支持，Python 数据科学生态体系得到了快速的发展，越来越多的数据工程师选择 Python 作为开发语言。然而，在实际工作中，许多工程师往往侧重于需求实现而忽视对业务的理解。本书针对这一盲区，根据不同的业务场景，从数据的角度梳理、思考问题，并有针对性地阐述了不同的策略、算法和案例。

在跟随本书学习的过程中，我们可以从全局上深入理解数据分析的精髓，并融会贯通——这对于初学者和初级数据工程师的能力提升尤为重要。

——阿橙，“Python 中文社区”公众号主编

数据分析是近年来的热点。几乎所有的互联网公司在产品上都告别了“拍脑袋”做决定的方式，而选择“用数据说话”。因此，也有越来越多的人投入到相关领域当中。Python 作为数据分析的重要语言，受到了广泛关注。然而，对于想要成为数据工程师的人来说，仅完成编程语言的学习是远远不够的。本书恰恰为这一阶段的学习者提供了很好的帮助：从数据分析的基本理论，到业内实践中的分析流程和常用工具，本书均做了较为完整的梳理。

除了理论讲解外，书中还附带了不少分析实例，便于读者理解和演练；此外，作者的行业经验保证了本书的实用性，为入行者指出了清晰的学习路径。

——Crossin，公众号“Crossin 的编程教室”作者、码课创始人

Python 语言继在 Web 大潮之下成为网站快速开发、服务端运维的明星语言之后，随着人工智能技术的飞速发展又迎来了新的一波高潮，成为人工智能领域的首选编程语言。

Python 语言易学易用，有丰富的数据处理包，社区也相当成熟，在数据工程师群体中是非常流行的语言。作为中国最早一批使用 Python 的人之一，看见 Python 逐步从一门小众语言变成推动技术进步的主流语言，很是欣慰。希望此书能够帮助有志于成为顶级数据工程师的朋友更好地掌握这门优秀的语言。

——洪强宁，爱因互动创始人兼 CTO

人工智能是当下最热门的技术领域之一，各大厂商紧锣密鼓进行战略布局：自动驾驶、个人助手、医疗健康、电商零售、金融、教育……如果把人工智能比喻成火箭，那么数据就是燃料。不管你是从事人工智能、机器学习，还是数据分析，都离不开数据，由此诞生了数据工程师的职业。

本书从数据分析、数据挖掘、深度学习等方面介绍了一名数据工程师应该掌握的数据工程的方法和数据分析的思路，书中总结的数学公式和代码实践让原来枯燥的概念变得有滋有味。有志于成为数据工程师的你，细细“品尝”本书，必有收获！

——刘志军，公众号“Python 之禅”

本书内容由浅入深，分别介绍了数据分析的常用工具、Python 在数据分析方面常用的包、如何运用 Python 做基础的统计分析和如何运用 Python 做数据建模……读完以后令人有一种从侏罗纪时代穿梭到未来时代的感觉，信息量很大。

更难得的是作者拥有工业界的背景，这使他可以从实践操作的角度，手把手教您打造一把数据分析的利剑。

一言以概之，本书没有繁杂的数学公式，只有挤不出水的干货。

——挖数，公众号“Washu66”

前　言

数据分析、数据挖掘、深度学习及云计算，是当前最热门的技术领域。1830年前后，Gauss、Legendre 等数学家奠基了数据分析的基础理论；1943 年，心理学家 Warren McCulloch 和数理逻辑学家 Walter Pitts 首次提出神经网络；19 世纪 80 年代，Hinton、Yann LeCun 等人提出 BP 算法及卷积神经网络；2006 年，深度置信网络研究成果发表。至此，数据建模理论研究的宏观大厦已初见雏形。

历史是如此的巧合，正当需要海量数据集和工程技术方案来处理数据时，云计算应运而生。2003 年，谷歌发表关于 Google File System、Google Bigtable 及 MapReduce 三篇论文，让大数据处理技术风靡全球。以此为基础，2010 年前后，整个云计算的概念及技术体系已经非常完善了。

数据理论的完善、工程技术的发展与无数创意的结合，使得 2010 年以后，整个人类社会进入了“数据时代”。无论是精细化运营，还是人工智能产品，对数据的应用无处不在；无论是政府机构，还是私有的大、中、小型企业，使用数据的热情都达到空前的高度。

2014 年，我加入百度公司，从事大数据处理及数据建模等相关工作。回首过

往，在该领域的几年中，我经历了云计算从雾里看花到如今的方兴未艾；人工智能的初现端倪到如今的高潮迭起。作为一名前线的数据工程师， 我深刻认识到，对我及大多数工程师而言,既无法像Jeff Dean等一样提出经典的大数据计算模型；也无法像Hinton、Yann LeCun一样提出具有深远影响的建模算法。我们所要做的，就是学习与汲取当前的理论与技术，结合应用领域，实现工程应用。这也是我写本书的初衷，希望能从宏观框架上梳理已有的数据分析理论与工程实施技术，并搭建相对系统的知识体系；同时，阐述工作实践中遇到的问题及解决的思路。

Python 简洁易懂的语法、丰富的类库、与大数据组件的无缝集成等诸多特点，使其成为数据工程师的首选编程语言。当然，只是掌握 Python 还完全不足以成为顶级数据工程师，因此，本书介绍数据处理知识体系，并以 Python 实现相关代码示例，力求让读者能使用 Python 完成数据处理的各个环节。

本书的第 1 章和第 2 章，简要说明了数据处理领域的基本概念，旨在让读者对数据处理工作有宏观的了解。第 3 章 ~ 第 5 章，主要讲述数据分析理论。笔者按照难易程度，将其划分成三个部分，即基础分析、数据挖掘和深度学习。第 6 章针对大数据分析，介绍了在工程实施过程中需要用到的工程组件和架构模式，并以一个具体的案例说明整个数据工程的实施流程。

本书适合以下读者阅读：① 对人工智能和云计算感兴趣的读者；② 刚进入数据处理领域的 IT 工程师；③ 希望从宏观上梳理数据处理知识体系的读者；④ 用 Excel、SPSS、Python 做过数据分析的数据分析师；⑤ 应用过 HDFS、Kafka 等大数据组件的 IT 工程师。

本书能够完稿，得益于外界诸多的帮助与指导。感谢数据领域的先驱者 Geoffrey Hinton、Yann LeCun、Jeff Dean 等，他们的著作是数据时代最重要的理论依据；感谢在百度工作中遇到杨振宇、李华青、王珉然、陈合等许多优秀的同事和领导，在和他们一起试错、交流的过程中，让我取得巨大的进步；感谢本书的编辑刘皎，在她不厌其烦地督促下，本书才从凌乱的只言片语中编辑成书；特别感谢女友孙万兴，在本书的撰写过程中给予的谅解与支持。

目　录

1　概述 1

1.1　何为数据工程师 1
1.2　数据分析的流程 3
1.3　数据分析的工具 11
1.4　大数据的思与辨 14

2　关于 Python 17

2.1　为什么是 Python 17
2.2　常用基础库 19
2.2.1　Numpy 19
2.2.2　Pandas 26
2.2.3　Scipy 37
2.2.4　Matplotlib 38

3　基础分析 43

3.1　场景分析与建模策略 43
3.1.1　统计量 43
3.1.2　概率分布 48

3.2 实例讲解 55
3.2.1 谁的成绩更优秀 55
3.2.2 应该库存多少水果 57

4 数据挖掘 60

4.1 场景分析与建模策略 60
4.1.1 分类 61
4.1.2 聚类 76
4.1.3 回归 86
4.1.4 关联规则 90
4.2 数据挖掘的重要概念 93
4.2.1 数据预处理 93
4.2.2 评估与验证 97
4.2.3 Bagging 与 Adaboost 99
4.2.4 梯度下降与牛顿法 102
4.3 实例讲解 105
4.3.1 信用卡欺诈监测 105
4.3.2 员工离职预判 110

5 深度学习 114

5.1 场景分析与建模策略 115
5.1.1 感知机 115
5.1.2 自编码器 119
5.1.3 限制玻尔兹曼机 123
5.1.4 深度信念神经网络 127
5.1.5 卷积神经网络 129
5.2 人工智能应用概况 137
5.2.1 深度学习的历史 137
5.2.2 人工智能的杰作 140
5.3 实例讲解 146
5.3.1 学习识别手写数字 146
5.3.2 让机器认识一只猫 151

6 大数据分析 160
6.1 常用组件介绍 160
6.1.1 数据传输 160
6.1.2 数据存储 165
6.1.3 数据计算 174
6.1.4 数据展示 180
6.2 大数据处理架构 188
6.2.1 Lambda 架构 189
6.2.2 Kappa 架构 192
6.2.3 ELK 架构 193
6.3 项目设计 194

参考文献 202

1 概述

首先，我们会从“软实力”与“硬实力”两个方面，介绍一名数据工程师应该具备的能力，并以“能力图谱”的方式列出数据工程领域的知识体系。本书正是围绕这些知识点，逐层细化讲解的。其次，本书会总结数据处理的一般流程：明确目标、确定方案、数据整理、建模分析、结果验证、总结展现，继而对各环节的具体工作进行详尽说明；并从易用性、适用领域等多个维度，介绍工作中常用的数据处理工具。最后，阐述笔者做大数据处理与分析中的一些思考，旨在让读者对大数据有更进一步的认识。

1.1 何为数据工程师

数据工程师无疑是大数据时代最热的名词之一。只要是从事数据相关工作的人员，都可以划分到该范畴。笔者认为“数据分析”与“数据工程”这两项能力，是数据工程师的核心能力。

“数据分析”能力，从实践的角度来看，就是工程师能根据数据给出分析建模的策略，解决业务中遇到的问题的能力。制定有效的建模策略，要求数据工程师

必须至少具备以下三类知识储备：统计分析、数据挖掘以及深度学习。本书的第 4~6 章分别讲述了这三个领域的相关知识。

“数据工程”能力，就是灵活运用数据处理组件及相关技术，实现数据分析中拟定策略的能力。数据处理的工程实施包含了“数据搜集”“数据传输”“数据存储”“数据计算”四个部分。这一系列流程的完成，要求数据工程师必须具备以下工程知识：消息队列、数据库技术、数据仓库、分布式文件系统和分布式计算平台等。在本书的最后一章，会对此做详细的介绍。

“数据分析”以及“数据工程”两项能力是数据工程师的硬实力。相比之下，对业务的理解则是与硬实力相辅相成的软实力。现实的工作中，一切技术手段最终都是为了业务的发展，理解这一点对数据工程师尤为重要。从初级数据工程师晋级为高级数据工程师，也是一个从被动实现到主动创新的过程。通常情况下，当面对上级或者同事提出的具体数据需求时，初级数据工程师一般处于被动实现的境地，这在初级阶段是无可厚非的；但是，对于高级数据工程师而言，则不能满足于被动地去实现需求，而要更进一步做到以下两点：① 应该主动去理解战略目标、产品方向和营销意图，围绕产品发展的各个阶段，综合当前业界已有的方法和思路，建立一套完善的数据支持体系；② 善于从数据的角度思考问题，保持数据敏感度。“啤酒-尿布”的故事正说明只有对数据敏感的人，才有可能从这种角度来发现数据间的关联。总之，一名优秀的数据工程师，不但要专心锤炼技术，更要着眼于现实的业务。

实践中，数据工程师不但要专注能力图谱（如图 1.1 所示）中的某几个点，还要在各个领域都有所涉猎。比如，合理地评估对海量数据集实施建模策略的可行性，就要求数据工程师不仅要了解模型构建，还必须具有工程实施的经验。所以，只有具备广而深的知识体系，才能掌控全局，更有效地提炼数据价值。

不可否认，数据工程师是一个更加需要经验的岗位。随着业务的深入发展，数据诉求会时刻发生变化；同时，不同业务之间的数据诉求也是天差地别的。笔者认为，拥有完善的分析体系和工程实施策略，并能针对具体场景给出完整解决方案的工程师，才能称为顶级数据工程师。

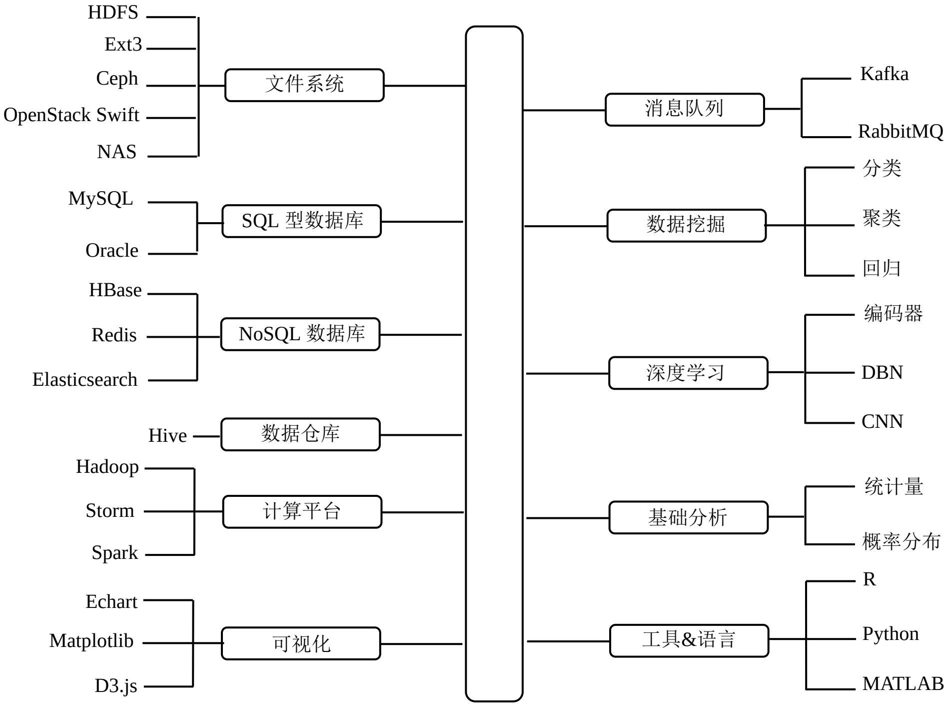


图 1.1　数据工程师能力图谱

1.2　数据分析的流程

与解决其他问题一样，数据分析也存在一般的模式化流程，通常可以划分为六个阶段。即明确目标、确定方案、数据整理、实施建模、结果验证和总结展现。

图 1.2 展示了数据分析的各个阶段。以下将详细说明各个阶段的具体实施思路，以及由“结果验证”阶段回退到“确定方案”阶段的原因。

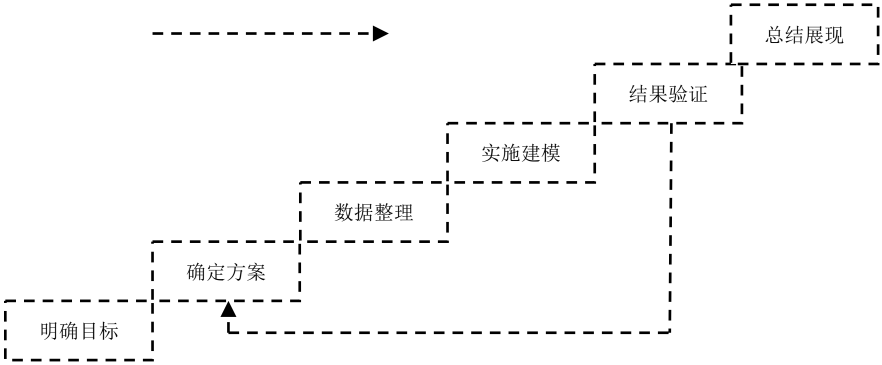


图 1.2　数据处理流程图

1．明确目标

明确且细化分析的目标，是数据分析中极为重要的一点，直接关系到全部工作是否能有效展开、业务是否能有收益。

从“分析目标”的维度，对数据分析做抽象归纳，可以把它分为三种类型，如图 1.3 所示：① 验证型分析；② 描述型分析；③ 预测型分析。

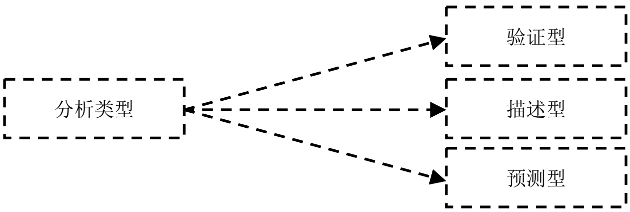


图 1.3　数据分析的三种类型

验证型分析，主要指对提出的问题进行数据验证。比如，某公司第二季度的销售额相比去年同期是否有所下滑；或者相比于选手 B，选手 A 的水平是否更加稳定等。提出问题的同时，用数据对其进行合理的验证，就是验证型的数据分析，也是日常分析中最常用的数据分析手段。

描述型分析，主要是从数据的角度说明现状或者问题。比如网站的运营状况可以通过网站的 PV、UV 或者留存率、转化率等数据指标进行描述。具体选择哪

些数据维度或者评价指标来描述数据？就笔者的经验而言，不同领域、不同事物的描述分析的维度虽然千头万绪，但都存在一定的、可以套用的模式。网站运营较为常用的指标有 PV、UV、二跳率、转化率、留存率等；网站运维常用的指标有页面平均响应时长、故障率、缓存命中率等；营销常用的有 STP 理论、SWOT 等。这些模式和领域密切相关，是业界同行经验的积累和总结，完全可以吸纳和应用，不必闭门造车。

预测型分析，主要指根据历史数据或者其他的数据信息，对可能发生或者即将发生的事情做出数据上的合理推测。比如，根据某地上年同期的降雨量预测今年同期的降雨量；或者通过 ARMA[1]模型预测股票大盘的走向；数据挖掘中回归决策树[2]对数据的分类，也是一种预测型数据分析。

2．确定方案

确定方案有三个步骤（参见图 1.4）：① 确认能否获取相关数据；② 选择可行的分析建模以及实施方法；③ 制定结果的校验准则。

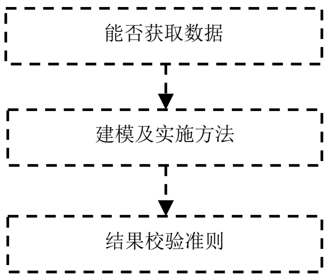


图 1.4　确定方案的三个步骤

能否获得数据决定了分析是否能够进行。在工程实践中，往往有很多可行的分析方法，然而能否获取相应的数据是一切工作的前提。一定要在已有数据的基础上，选择与拟定建模方法，以防止后续工作的徒劳无功。

建模的方法，由简单到复杂可以划分为三类（参见图 1.5）：① 基础分析 ② 数据挖掘 ③ 深度学习。这也是本书后续章节的叙述组织方式，虽不能大而全地呈现数据分析的细节，但是基本描绘了整体的知识框架。

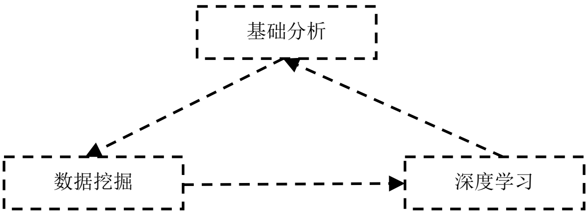


图 1.5　建模的方法

基础分析主要解答事物的统计特征，以及概率的相关问题。它首先研究是否可以通过均值、方差等简单的统计量来说明问题；其次，分析数据是否符合某种分布，如果能给出数据的有效分布，就可以合理地计算事物的概率。这些简单的分析在某些场景下具有事半功倍的效果。数据挖掘主要解决分类、聚类、关联的相关问题。如果在分析中遇到这几类问题，可以尝试从数据挖掘的角度来寻求方法。深度学习和数据挖掘类似，也是用来研究分类和识别的问题。它们之间的最大区别是：前者能自动提取数据的特征，并对非线性数据集具有良好的效果，有些文献把数据挖掘称为浅层学习。深度学习常用在图像识别以及声音识别的场景中。本书的第 5 章和第 6 章将分别对两者进行更为详细的讲述。

一般情况下，应由易到难地选择建模方式，解决实际问题。比如针对一组数据，我们首先要考虑基本的统计量以及概率分布是否能达到数据分析的目标；其次，思考能否运用数据挖掘的方法来对数据做进一步的分析；最后探讨深度学习的思路能否更好地解决问题。总之，兵无常势，水无常形，在具体问题的基础上，只有灵活运用多种建模手段，才能更好地达到分析的目标。

最后，制定结果的校验准则对数据分析尤为重要。在实践中，统计数据的对错往往很难被发现和评估；同时，错误的统计结果或者分析结论在某些时候可能会造成巨大的损失。因此，制定完善的数据校验策略来验证数据分析结果的可信度是极其重要的。本书会在数据校验部分对做校验的一些经验性方法进行简单的阐述。

3．数据整理

数据搜集与整理的难处有两点：① 原始数据的渠道来源多样、格式繁杂；② 数据的分析必然和业务紧密相关，要做好数据工作，必须以理解业务为前提。

以笔者参与的某项目为例，为了统计渠道商数，会涉及网站 PV（PageView）、系统响应时间等指标，后端数据源涉及 Nginx 日志，MongoDB 数据库用户数据，以及其他部门 FTP 提供的文件等。并且，还需要依据文档以及业务流程，分门别类地进行数据整合，并录入数据仓库，以供后续的数据分析使用。

原始数据的组织管理是否条理清晰会直接影响后续的分析建模。所以，一定要认真严肃地对待数据搜集与整理这一环节。

图 1.6 中展示了数据搜集的一般过程。其中 ETL 是 Extract-Transform-Load 的缩写，其功能是从源端抽取（Extract）原始数据并进行格式化等转换（Transform），最终加载（Load）到数据仓库。ETL 是数据处理中最基础的概念，这也表明了它在数据处理流程中的重要性[3]。从业务的角度出发，解耦 ETL 功能模块，并合理地构建数据仓库直接决定了后续的数据建模与应用。

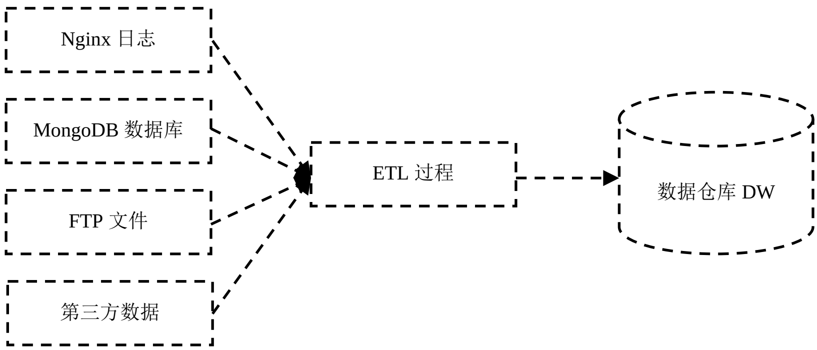


图 1.6 数据整理的过程

4．实施建模

逐步实施已确定的建模方案只是水到渠成的事情。本章第 3 节介绍的多种数

据建模工具是帮助数据工程师高效完成数据工作的利器。这些工具已经包含了主流的数据建模方法，在掌握一定知识的基础上，完全可以将其当成“黑盒子”来使用。需要特别指出的是，这些工具并不能完全胜任大数据的处理工作，本书将在大数据处理的章节详细讲述如何高效处理海量数据。

5．结果验证

结果验证是数据分析中最重要的一环。有些结果较容易验证，比如预测用户是否购买，可以通过预测结果和实际是否购买做对比；但是有些统计数据难以评估其是否正确。比如，通过历史日志统计分析得出当日的网站 PV 是 10,000，那么这个数值的正确性是很难衡量的——日志传输过程中的丢失，或者统计的 BUG 都可能导致该值存在偏差，但又很难被发现。

以下介绍笔者常用的两种结果验证的方法：① 多维对比法；② 冗余验证法。

多维对比法是指对于某一项数据，从线和面的维度进行对比。从线的维度，对比该指标今天的数据和昨天的数据，或过去一周的数据是否有较大的波动或者异常；从面的维度，对比该指标的统计结果与本次统计的其他数据是否存在冲突。比如某网站今天搞营销活动，用户数量增加了两倍，但是请求数却没有多大变化，则此时可能存在异常。

冗余验证法是针对某些重要指标、或能直接影响公司战略决定的指标，请多位分析师做同一项数据统计，或者用不同的方式统计相同的指标，并对比结果的差异，找出统计中可能存在的问题。

6．总结展现

数据和数字都是高度抽象化的概念，仅仅以数字的方式呈现出数据分析的结果，难以直观地说明问题。相比之下，图形可以合理而直观地展现数据，能更好地展现和说明分析的成果。根据不同的场景和需求，可以用不同的图形来可视化数据。数据可视化是数据分析中不可或缺的一环。

最常用的几种图形分别为：① 柱状图 ② 折线图 ③ 饼图 ④ 散点图 ⑤ 漏

斗模型图 ⑥ 雷达图；当然在某些特定场景中还有股价图、圆环图、气泡图等。下面我们以某网络商城的手表销售为例，说明各种图形的应用场景。

图 1.7 为商场从 1 月到 12 月期间，某款手表每月的销售量。由该图可以看出，由于口碑或者营销策略的影响，手表销售量逐月攀升。和其他图形相比，折线图能更好地展现数据的波动、趋势等场景。

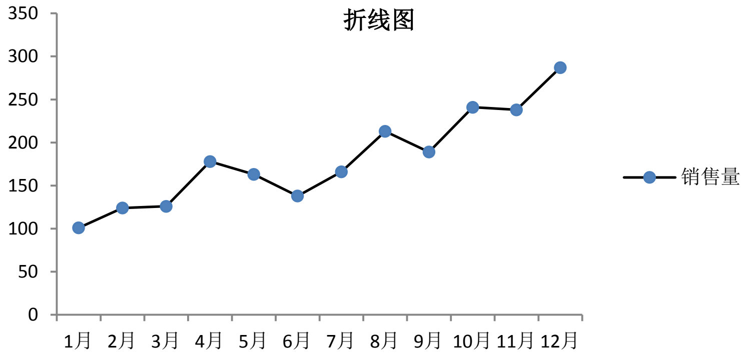


图 1.7　手表的月销售量

图 1.8 为商场 1 月到 12 月期间手表销售的柱状图。折线图反映趋势的变化，柱状图能更好地展现量的变化。

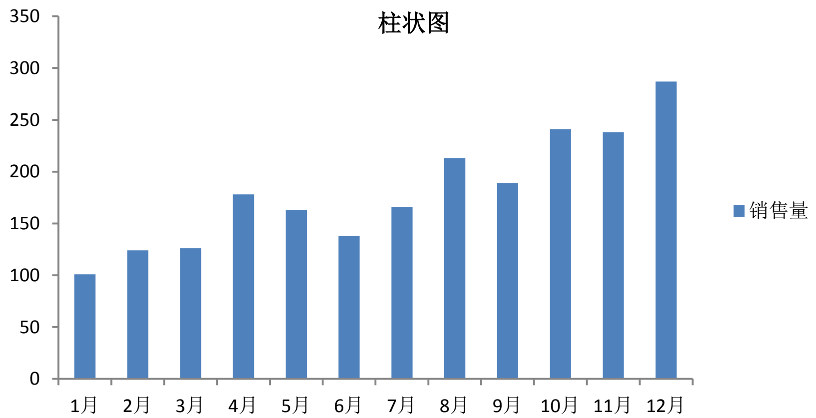


图 1.8　商场 12 个月的手表销售量

图 1.9 为 1 月到 12 月期间，每月手表销售量占全年的比例。从中可以看出，饼状图更加适用于凸显比例的统计场景中。

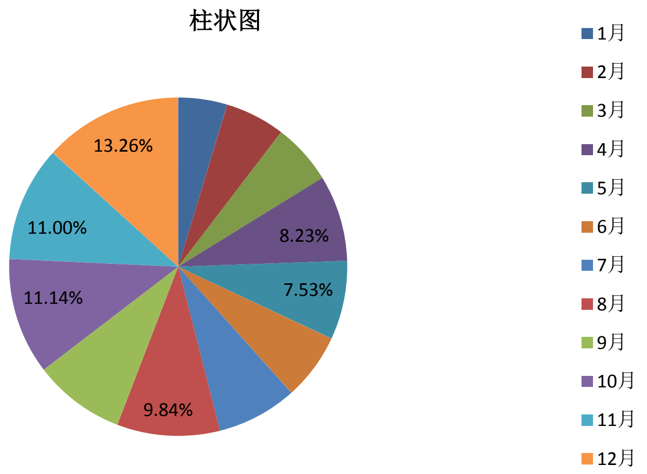


图 1.9　商场手表月销量的全年占比

某手表有 A 款、B 款、C 款、D 款、E 款、F 款、G 款等七种型号，现在统计顾客对每种型号打分的历史数据，并绘制散点图，如图 1.10 所示。从中可以看出各种型号手表的评分分布，从而了解顾客对每种型号手表的整体评价情况。因此散点图对于需要展现各种统计分布的场景较为有效。

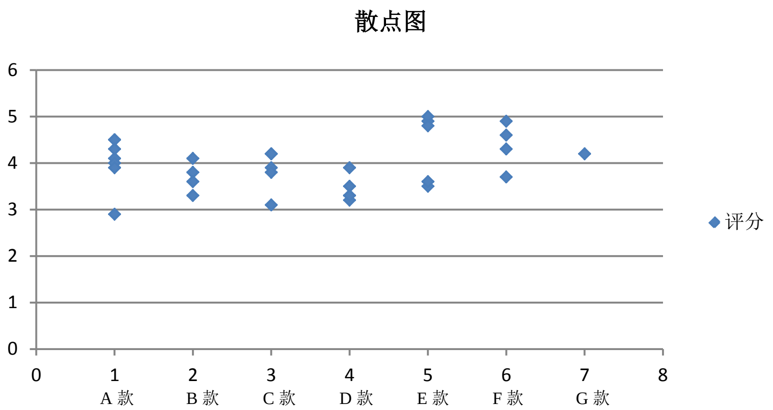


图 1.10　商场手表顾客打分散点图

我们再依据“浏览次数”“好评次数”“询问次数”“售卖个数”四个维度，分别对 A、B 两款手表绘制雷达图，如图 1.11 所示。该图简明地展现出 B 款的“浏览次数”与 A 款相近，“询问次数”大于 A 款；但是 B 款的“售卖个数”和“好评次数”小于 A 款。由此说明 B 款在外观上更有吸引力，很有可能是功能或者细节上的问题导致其售卖和好评率低于 A 款手表。在需要多维数据对比的场景中，雷达图更为适用。

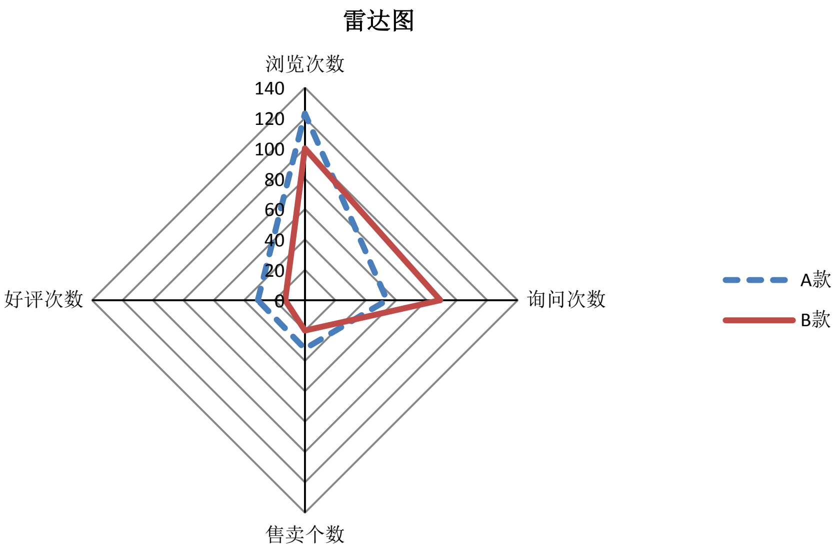


图 1.11 商场手表月售卖统计雷达图

数据在分析处理各个环节的流转过程，是一种十分抽象并极易出错的过程。数据可视化的目的就是为了更直观地展现数据的含义，从而让工程师更好地理解数据，提炼价值，降低出错的概率。根据不同的场景灵活运用各种图形呈现数据，能有事半功倍的效果。

1.3 数据分析的工具

基于不同的应用领域，在数理统计的理论基础上，各机构和公司推出了多款高可用的数据分析工具。本节从易用性、专业性以及应用场景等维度，着重介绍

MATLAB、SPSS、Stata、SAS、EViews、Excel、Python、R 这几款工具。

（1）MATLAB

MATLAB 是 Matrix Laboratory（矩阵实验室）的缩写，是一款由美国 The MathWorks 公司出品的商业数学软件。MATLAB 不仅仅是一款可以用来做统计分析的软件，它还可以高效地处理其他很多的数学问题[4]。它常被用于各种数学建模和工程设计，相比于它强大的统计分析功能，这可说是大材小用。它具有丰富的库函数（工具箱）；内嵌绘图功能，可实现数据的多维度展现；同时有良好的交互设计，活跃的社区以及丰富的文档……这些都使它具有极高的易用性，我们也可使用解释执行语言对其进行编程。

（2）SPSS

SPSS 是 Statistical Product and Service Solutions 的缩写，是一款由 IBM 公司推出的用于分析运算、数据挖掘、预测分析和决策支持等一系列任务的软件产品及相关服务的总称[5]。SPSS 可以用在经济分析、市场调研、自然科学等林林总总的领域。它最大的特点是“简单易用”。虽然它对前沿理论的支持不够全面，但是囊括了绝大部分常用的统计方法。简单的操作方式、友好的操作界面，再加上强大的功能，使其在国内统计分析工作领域吸引了大量用户。

（3）Stata

Stata 是 Statacorp 于 1985 年开发出来的统计程序[6]。和 SPSS 一样，它也支持常用分析方法，可用于多个领域，不过实践中在医学和生物学研究上的应用较多。Stata 采用菜单和编程相结合的使用方式，其易用性虽不如 SPSS，但在功能上略胜一筹。它在企业和学术机构的应用比较广泛。

（4）SAS

SAS 诞生于北卡罗莱纳州立大学，起初只是一个用于分析农业研究的项目。随着需求的增长，它的使用范围扩展至医药企业、银行业以及学术和政府机关[7]。

SAS 系统提供的主要分析功能包括统计分析、经济计量分析、时间序列分析、决策分析、财务分析和全面质量管理工具等。SAS 功能极其强大，算法包非常完善，但是它是纯编程界面，易用性低且入门困难，适合高级数据分析师或者专业人士使用。在统计分析领域，SAS 一度是“统计分析系统”的缩写，被誉为国际上的标准软件和最具权威性的优秀统计软件包。

（5）EViews

EViews 是 Econometrics Views 的缩写，由 Quantitative MicroSoftware（QMS）开发，是一款基于 Windows 设计的统计分析软件[8]。EViews 可以用于常规的统计分析，但它在计量经济分析方面特别有效。它的易用性高，且相比于上述其他分析软件，入门级别低。针对计量经济学相关的分析，可以首先考虑该软件。

（6）Excel

Excel 是微软公司为 Windows 操作系统编写的一款电子表格系统，可以画各种图表、做方差分析、回归分析等基础分析。它的专业性虽然不高，但是完全可以胜任日常工作中简单的统计分析工作。同时，它极其方便的操作方式，以及 Microsoft Office 软件包成员之一的身份，使它成为最流行的个人计算机数据处理软件。

（7）Python

Python 是由荷兰人 Guido van Rossum 于 1989 年发明的一种面向对象的解释型编程语言，并于 1991 年公开发行第一个版本[9]。Python 是本书各种代码实现所使用的语言。之所以把 Python 语言列为数据分析的工具，是因为围绕它实现的各种数据分析与数据可视化的开源代码库被广泛应用。同时，Excel、SPSS 等工具虽然具有可操作的界面，但并不能有效地结合 Hadoop、Hive 等组件有效地处理海量数据，而这些都是 Python 可以胜任的。

（8）R 语言

R 是专用于统计分析以及可视化的语言，是 AT&T 研发 S 语言时的产物，可以认为是 S 语言的另一种实现方式[9]。同 Python 一样，R 也提供了极其丰富的库函数来做统计和展现。因为 R 太过强大且拥有大量的用户，为了能顺应用户的习惯，降低学习的成本，Python 在数据处理上的很多库函数都是模仿 R 的实现，以保持与其基本一致的使用方式。

我们下面通过表 1-1 对比上述八款软件。

表 1-1　八款软件对比表格

工具名称	公司	是否免费	易用性	专业性	可编程	常用场景及领域
MATLAB	MathWorks	否	中	高	是	统计分析只是应用的一个方向，适合数据处理以及工程建模的各个领域
SPSS	IBM 公司	否	高	中	是	统计分析专业人士的入门级软件。可用于经济分析，市场调研等社会科学各个领域
Stata	Stata 公司	否	中	中	是	统计分析专业人士的进阶级软件。多用于医学，生物学研究领域
SAS	SAS 公司	否	低	高	是	统计分析专业人员的殿堂级软件。可应用于各个统计分析领域，是高级分析人员更青睐的统计分析利器
EViews	QMS 公司	否	高	中	是	能很好地处理时间序列分析等相关问题，主要应用于计量统计学领域
Excel	微软	否	高	低	是	非专业人士使用的简单统计分析软件。可以胜任日常工作中简单的数据统计、数据整理以及数据展示的工作
Python	-	是	中	高	是	完全的编程实现。可用于任何领域，并且与大数据组件结合可以方便地处理海量数据
R	AT&T	是	中	高	是	同 Python

1.4　大数据的思与辨

我们可以从各个维度提炼和概括大数据的多种特点。IBM 曾总结出大数据的

5V 特性，即 Volume（容量）、Velocity（高速）、Variety（多样）、Value（价值）、Veracity（真实性）[10]。不过不管如何定义大数据，始终逃离不了“数据量大”这一最根本的特征。

大数据的大到底有多大？下面给各位读者一个量化的感受：2010 年左右普通 PC 机的硬盘容量大概在 512GB~1TB，到了 2016 年，硬盘容量已经达到 TB 级别，天猫或者京东上售卖的硬盘容量普遍在 1TB~3TB。以笔者参与的某项目为例，Nginx 服务器每天产生的日志多达约 500TB，每个星期大约有 3~4PB 的数据量。为了实现特定的业务分析需求，该类日志数据需要存储 6 个月，也就是最多的时候有 90PB 的数据需要分析和处理。

因此，相比于“小数据”分析立足于如何使用更有效的理论和模型；“大数据”的分析则要先解决两个问题：如何存储和如何计算，最后才是如何分析。

需求推动技术的发展，技术促进需求的实现。为了解决海量数据的存储问题，2000 年到 2010 年期间以谷歌、Facebook、百度等巨头公司为代表，掀起了一股云计算的热潮。云计算用数以万计的普通机器，组成大规模集群，解决巨大数据量的存储和计算问题。大浪淘沙之后，在 2010 年左右，云计算的概念被广为接受，并最终被分成 IaaS（基础设施即服务）、PaaS（平台即服务）、SaaS（软件即服务）三种技术体系。IaaS 解决大规模集群的管理，利用虚拟化技术在一台机器上虚拟多台机器，实现资源的合理利用。IaaS 服务商主要对外提供虚拟网络、虚拟机等基础设施服务，比如亚马逊的 AWS 云虚拟机，阿里云等。PaaS 处于 IaaS 层之上，使用 IaaS 提供的基础设施，构建分布式文件系统以及分布式计算平台。这类服务比较有代表性的有谷歌的 GFS 文件系统，Apache 的 Hadoop 系统等。SaaS 则处于 PaaS 之上，利用 PaaS 提供的存储及计算平台实现丰富多样的应用，比如百度网盘，亚马逊商品的即时推荐等。

关于存储，日志数据、文件数据等不可修改的数据，常用 Apache 的 HDFS 存储；网盘类数据则较常用 OpenStack 的 Ceph 存储。不管用哪种组件，其目标都是在“能存储”的前提下，提升各种场景下的 I/O 速度。

关于计算，离线计算最常用的计算模型是 MapReduce。基于该模型，Apache 实现了 Hadoop 分布式系统（HDFS 是其一部分），适用于离线分析；同时，流式计算也有一些代表的组件，如 Storm、Spark Streaming 等。

关于分析，大数据的分析理论以及建模思路与“小数据”分析并无差别，但它需要结合存储与计算的组件，来决定分析是否可实施。举一个最简单的例子，假定有 1MB 的数据，数据格式为每行记录一个数值，求该数据中最大的记录值。求解的过程十分简单：读取文件数据，对比获取最大。但是，假如有 10PB 的数，那又该如何计算？单台机器已无法存储这样的数据，更不可能简单地全部读取一遍。这时，就需要综合使用上面提到的所有技术。首先，数据不断累积，并被录入 HDFS 文件系统；其次，使用 Hadoop 平台的 MapReduce 计算模型，统计集群各个存储节点的局部最大值；在经过多次 MapReduce 过程之后，可以把数据量缩小到单台机器可以存储的范围，最终计算出全局最大值。从此例中可以看出，“大数据”分析的理论依然属于统计分析领域的理论和知识范畴，只是在实施的过程中，需要灵活运用大数据的技术来完成。同时，由于受计算模式、存储方式的限制，很多数据分析的方法，无法有效地应用在“大数据”上。

大数据时代，数据的分析不再着眼于抽样，而是用全量取代了样本[11]。同时，工业界和学术界也在研究和探讨更加便利的大数据处理方案，以便更多的理论和策略能够在海量数据集上运作。笔者以为，在大数据存储和计算较为完善的今天，有效地运用数据分析、数据挖掘和深度学习，创造更神奇的数字化世界、人工智能世界是大势所趋。

2 关于 Python

Python 是数据分析领域极受欢迎的主流程序语言。为什么要用 Python，每位读者见仁见智。然而，无论哪种工具，都有一定的适用范围。本章第 1 节，将以多维对比的方式，说明在什么场景下使用 Python 更为合理。

在 Python 的基础库部分，将着重讲述 Numpy、Pandas、Scipy 以及 Matplotlib 的常规用法，为后续的示例程序打下基础。但本书并不希望成为查阅式的手册，因此在内容的安排上将力求厘清整体框架，配以部分示例，做简明扼要的讲解；一些晦涩的语法或者使用频率不高的特性不会在此赘述，读者可参阅官网[12]自行深入学习。

2.1 为什么是 Python

数据分析的工具可以分为两类：一是以 MATLAB、SPSS、Excel 等为代表的具有界面化操作，且可编程辅助的软件式工具，一是以 Python、R、Java 等为代表的纯编程分析的程序语言式工具。

（1）软件式工具和语言式工具在软件分析领域的优缺点

软件式工具不太适合需要例行化自动分析的问题。比如 Nginx 每小时产生一批数据，需要统计其中 HTTP 状态码为 200 的请求数以及流量峰值等数据。在这种情况下用 Python、R 或者 Java 能够很好地实现例行化统计；然而，用 Excel 或者 SPSS 实现则非常困难，当然 Excel 加上 VBA 也可以实现，但是十分不方便。

软件式工具不太适合大数据处理的问题。对于亿行甚至更多的数据，使用软件式工具分析，其效率极低或者根本无法操作。但是，Python 和 R 等语言式工具，能够通过结合大数据组件，实现对海量数据的有效处理，而且一系列的过程皆可用编程自动化完成。

所以，数据量小，非例行化的分析使用软件式工具可能更加方便；数据量大，或者需要例行化分析输出的场景，选择一门语言式工具进行分析更为高效。

（2）R、Python、Java 等语言式分析工具的优缺点

Python、R 等属于解释型语言，简单的说就是编写好程序，直接运行，不需要编译链接等各种过程。Java、C++等属于编译型语言，需要对代码进行编译链接等一系列操作，之后才能形成最终的可执行程序。

在数据分析的场景中，解释型语言有诸多好处。

① 不需要编译链接。编译链接的过程是极容易出错的，尤其对于编程能力较弱的数据分析师而言，根本无法解决编译链接中的各种问题。因此，选择解释型语言，其好处不言而喻。

② 相比于 C++、Java 等编译型语言，Python、R 等解释型语言语法和结构相对简单，对于专注于数据分析的新手，一般学习一周左右就可以上手工作了。

③ Python、R 拥有和数据分析相关的大量开源库和分析框架，可直接使用，非常方便。

综上，在选择语言式分析工具时，Python、R 会是更好的选择。但是，到底

用 Python 还是 R 呢？如果只针对数据分析，那么 Python、R 确实没有大的差别；但是，R 仅仅在数据分析领域比较成熟，而 Python 则可以应用到除数据分析以外的几乎所有的程序开发领域。比如，Python 可以使用各种开源库简易地实现 Web 服务端、爬虫等程序；同时，其多样的“语法糖”可以方便地完成复杂的运算和多种设计模式。由于数据分析在多数场景下并不是孤立的，可能需要先完成爬取数据、或者先保存服务端日志等工作，在这种情况下，为什么不用 Python 呢？

在大数据时代的浪潮中，数据工程师承担着分析和工程的双重角色。他们不仅要掌握数据分析的方法，还需要实现数据的搜集、过滤、存储，并在此基础上分析、挖掘数据的价值，因此，在这样的大背景之下，与当前多种主流的数据分析工具相比，Python 具有绝对的优势。

2.2　常用基础库

2.2.1　Numpy

Numpy 是 Python 的高性能科学计算基础库，基于 ndarray（n-dimensional array object，一种多维数据对象，实现了多种操作和数学运算）。该对象与 Python 提供的 list 数据结构十分类似，但是 list 存储的是数据指针，而 ndarray 存储的是数值。同时，Python 也提供 array 来存储数值，但 array 是一维数组。引入 ndarray 并配合 ufucn 函数可以更高效地实现数据处理。本节围绕 ndarray 的创建、存取、选择、函数运算以及 I/O 操作举例说明，让读者对 Numpy 有直观的认知，表 2-1 是 Numpy 的内容概览。

官网手册

https://docs.scipy.org/doc/numpy/reference/

表 2-1　Numpy 内容概览

数据结构	说　　明	相关操作
ndarray	多维数组，支持直接存储数据的值。	1. 创建 2. shape 及 reshape 3. 存取 4. ufunc 函数 5. 文件 I/O

示例版本

版本号：1.11 版本

日期：2016.5.29

示例讲解

（1）引入包

```
>>>import numpy as np
```

（2）数组的创建

```
>>>a = np.array([1, 2, 3, 4, 5], dtype=np.int32)
>>>print a
[1 2 3 4 5]
>>>b = np.array([[1, 2],[3, 4], [5, 6]], dtype=np.int32)
>>>print b
[[1 2]
[3 4]
[5 6]]
>>>#通过 Numpy 提供的函数来创建数组
>>>#arange 函数创建数组
>>>a = np.arange(1, 10, 1)
>>>print a
[1 2 3 4 5 6 7 8 9]
>>>print type(a)
<type 'numpy.ndarray'>
>>>print a.dtype
int64
>>>#linspace 函数创建数组。指定开始值、终止值以及元素个数来创建一维数组，该数组各个值成等差数列
>>>a = np.linspace(1, 10, 20)
```

```
>>>print a
[  1.          1.47368421   1.94736842   2.42105263   2.89473684
   3.36842105   3.84210526   4.31578947   4.78947368   5.26315789
   5.73684211   6.21052632   6.68421053   7.15789474   7.63157895
   8.10526316   8.57894737   9.05263158   9.52631579  10.        ]
>>>print a.dtype
float64
>>>#logspace 函数创建数组。指定开始值、终止值以及元素个数来创建一维数组，该数组各个值成等比数列
>>>a = np.logspace(1, 10, 8)
>>>print a
[ 1.00000000e+01  1.93069773e+02   3.72759372e+03   7.19685673e+04
   1.38949549e+062.68269580e+07                     5.17947468e+08
1.00000000e+10]
>>>print a.dtype
float64
>>>#fromfunction 函数创建数组。该函数的第一个参数为计算数据元素的函数；第二个参数是代表数组的大小的序列，序列的每一个值代表数组对应维度的大小
>>>def funcA(i):
...     returni + i
>>>a = np.fromfunction(funcA, (10, ))
>>>print a
[  0.   2.   4.   6.   8.  10.  12.  14.  16.  18.]
>>>print a.dtype
float64
>>>def funcB(i, j):
...     returni + j
...
>>>b = np.fromfunction(funcB, (3, 3))
>>>print b
[[ 0.  1.  2.]
[ 1.  2.  3.]
[ 2.  3.  4.]]
>>>print b.dtype
float64
```

（3）shape 与 reshape 函数

```
>>> #内容：shape 与 reshape 函数
>>>a = np.array([1, 2, 3, 4, 5])
>>>#查看数组大小
>>>print a.shape
(5,)
>>>b = np.array([[1, 2, 3], [4, 5, 6]])
```

```
>>>print b.shape
(2, 3)
>>>#调用 reshape 改变数组的重新排列数组维度，同时，保存数组的大小不变
>>>a = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
>>>print a.shape
(10,)
>>>b = a.reshape(2, 5)
>>>print b
[[ 1  2  3  4  5]
[ 6  7  8  9 10]]
>>>a.reshape(-1, 2)
[[ 1  2]
[ 3  4]
[ 5  6]
[ 7  8]
[ 9 10]]
>>>a.reshape(2, 2)
Traceback (most recent call last):
File "<stdin>", line 1, in<module>
ValueError: total size of new array must be unchanged
```

（4）获取 ndarray 数组元素

```
>>> #内容：数组存取
>>>#下标存取
>>>a = np.arange(1, 10, 1)
>>>print a
[1 2 3 4 5 6 7 8 9]
>>>a[0]
1
>>>a[0:]
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
>>>a[:-1]
array([1, 2, 3, 4, 5, 6, 7, 8])
>>>a[1:8:2]
array([2, 4, 6, 8])
>>>a[::-1]
array([9, 8, 7, 6, 5, 4, 3, 2, 1])
>>>  #同时，根据下标存储获取的数组与原数组共享地址空间。示例如下
>>>a = np.arange(1, 10, 1)
>>>print a
[1 2 3 4 5 6 7 8 9]
>>>b = a[0:3]
>>>print b
```

```
[1 2 3]
>>>b[1] = -10
>>>print b
[  1 -10   3]
>>>printa
[  1 -10   3   4   5   6   7   8   9]
>>> #使用整数序列作为数据下标，从而获取数组元素。与使用下标获取数组不同的是，其结果不与原数组共享地址空间
>>>a = np.arange(1, 10, 1)
>>>print a
[1 2 3 4 5 6 7 8 9]
>>>b = a[[1, 2, 5]]
>>>print b
[2 3 6]
>>>b[0] = 100
>>>print b
[100   3   6]
>>>print a
[1 2 3 4 5 6 7 8 9]
>>>#使用布尔数组作为数据下标，从而获取数组元素
>>>a = np.arange(1, 5, 1)
>>>print  a
[1 2 3 4]
>>>a> 2
array([False, False,  True,  True], dtype=bool)
>>>a[a> 2]
array([3, 4])
```

（5）ufunc 函数

ufunc 的全称是 universal function，即通用处理函数。Numpy 提供的 ufunc 函数大致可以划分为五类，分别为：Math operations（数学操作）、Trigonometric functions（三角函数）、Bit-twiddling functions（位操作）、Comparison functions（比较函数）以及 Floating functions（浮点函数）。下面分别对各种类型的部分函数给出使用示例。

```
>>> #内容：数学函数
>>> #数学操作包含 add、subtract、multiply、divide，log、sqrt 等一系列函数
>>>a = np.array([1, 2, 3, 4, 5])
>>>b = np.array([6, 7, 8, 9, 10])
>>>np.add(a, b)
array([ 7,  9, 11, 13, 15])
```

```
>>>np.subtract(b, a)
array([5, 5, 5, 5, 5])
>>>a + b
array([ 7,  9, 11, 13, 15])
>>>b - a
array([5, 5, 5, 5, 5])
>>>np.log10(100)
2.0
>>>  #如果先让结果覆盖原有数组，可以参考如下示例中的方式使用
>>>a = np.array([1, 2, 3], dtype=float)
>>>b = np.sqrt(a)
>>>print a
[ 1.  2.  3.]
>>>print b
[ 1.          1.41421356  1.73205081]
>>>c = np.sqrt(a, a)
>>>print a
[ 1.          1.41421356  1.73205081]
>>>print c
[ 1.          1.41421356  1.73205081]
>>>id(a) == id(c)
True
>>> #内容：三角函数
>>> #包含 sin、cos、tan 等一些列函数。需要特别说明的是其参数并不直接代表三角函数的度数，而是使用 pi 替代。其中 2pi 相当于 360 度，所以 np.sin(np.pi) = 1。>>>np.sin(np.pi / 2)
1.0
>>>np.sin(np.array((0.,30.,45.,60.,90.))*np.pi/180.)
array([ 0., 0.5, 0.70710678, 0.8660254, 1.])
>>>np.cos(np.array([0, np.pi/2, np.pi]))
array([  1.00000000e+00,   6.12323400e-17,  -1.00000000e+00])
>>> #内容：位操作函数
>>> #包含按位的与、或、异或、左移、右移等操作
>>>np.binary_repr(5)
'101'
>>>np.binary_repr(7)
'111'
>>>np.bitwise_and(5, 7)
5
>>>np.bitwise_or(5, 7)
7
>>>np.left_shift(5, 2)
20
>>>np.right_shift(5, 2)
1
```

```
>>> #内容：比较函数
>>> #包含了数值比较、逻辑运算、取最大值、取最小值等操作
>>>np.greater([1, 5],[2, 10])
array([False, False], dtype=bool)
>>>np.greater_equal([4, 2, 1], [2, 2, 2])
array([ True,  True, False], dtype=bool)
>>>x = np.arange(5)
>>>np.logical_and(x>1, x<4)
array([False, False,  True,  True, False], dtype=bool)
>>> #内容：浮点函数
>>> #包含了类型判别、是否是最大值、向上取整等操作
>>>np.iscomplex([3+1j, 6+0j, 7.5, 6, 2j])
array([ True, False, False, False,  True], dtype=bool)
>>>np.isinf(np.inf)
True
>>>a = np.array([2.1, -2.1, 3.54, 7,2, -7.2])
>>>np.ceil(a)
array([ 3., -2.,  4.,  7.,  2., -7.])
>>>np.floor(a)
array([ 2., -3.,  3.,  7.,  2., -8.])
```

（6）文件 I/O

Numpy 提供一些文件存取操作，方便将 ndarray 数组的元素存储到文件以及从文件读取数据来初始化数组。其中 tofile、fromfile 以二进制形式的方式存取；save、load 函数以 Numpy 的专有格式存取；savez 可以保存多个数组到同一个文件，并可以通过 load 循环取出。tofile，fomfile 示例如下。

```
>>>a = np.array([1, 2, 3, 4, 5])
>>>print a.dtype
int64
>>> #tofile、fromfile 示例
>>>a.tofile("a.bin")
>>>b = np.fromfile("a.bin", dtype = np.int64)
>>>print b
[1 2 3 4 5]
>>> #save、load 示例
>>>np.save("a.npy", a)
>>>b= np.load( "a.npy" )
>>>print b
[1 2 3 4 5]
>>> # savez、load 示例
```

```
>>>a = np.array([[1, 2, 3], [4, 5, 6]])
>>>b = np.array([1, 2, 3])
>>>c = np.array([1.2, 3.1, 3.3])
>>>np.savez("data.npz", a, b, c)
>>>arr = np.load("data.npz")
>>>arr["arr_0"]
array([[1, 2, 3],
      [4, 5, 6]])
>>>arr["arr_1"]
array([1, 2, 3])
>>>arr["arr_2"]
array([ 1.2,  3.1,  3.3])
```

2.2.2　Pandas

Pandas 是基于 Numpy 构建的高性能数据统计库，其内容如表 2-2 所示。它提供了 Series、DataFrame、Panel 三种数据结构，并在此数据结构的基础上提供创建、存取、统计、缺失值处理以及 I/O 输入输出等函数操作。Panel 衍生于计量经济学的分析中，使用的频率并不是很高。因此本节着重围绕 Series 和 DataFrame 来介绍 Pandas 库。

官网手册

http://pandas.pydata.org/pandas-docs/stable/

内容概览

表 2-2　Pandas 内容概览

<table>
<tr><th>数据结构</th><th>说　　明</th><th>相关操作</th></tr>
<tr><td>Series</td><td>一维数组，带有标签作为索引</td><td rowspan="4">1. 数据创建
2. 数据查看
3. 数据选择
4. 数据设置
5. 缺失值处理
6. 汇总与统计
7. 类 SQL 操作
8. 时间序列
9. 文件 I/O</td></tr>
<tr><td>DataFrame</td><td>二维数组，与 SQL 的表结构类似。可以理解成 Series 为 Value 的 Map 数据结构</td></tr>
<tr><td>Panel</td><td>三维数组，可以理解成 DataFrame 为 Value 的 Map 数据结构</td></tr>
<tr><td>Panel4D\PanelND</td><td>在版本 0.19.0 中已过时、后续会逐渐从 Pandas 包中移除</td></tr>
</table>

示例版本

版本号：0.19.1

日期：2015.11.03

示例讲解

（1）引入包

```
>>>import numpy as np
>>>import pandas as pd
```

（2）创建对象

```
>>>#================创建 Series 对象================
>>> #传入 list 创建 Series，并生成默认的整形索引
>>>s = pd.Series([1, 2, 3, 4, 5]) >>>prints
0    1
1    2
2    3
3    4
4    5
dtype: int64
>>>s = pd.Series([1, 2, 3, 4, 5], index=["a", "b", "c", "d", "e"])
#指定索引创建 Series
>>>prints
a    1
b    2
c    3
d    4
e    5
dtype: int64
>>>#传入 Numpy 的 ndarray 数组创建
>>>pd.Series(np.arange(1, 5, 1))0    1
1    2
2    3
3    4
dtype: int64
>>>#================创建 DataFrame 对象================
...
>>>dates = pd.date_range('20170101', periods=5)
>>>df = pd.DataFrame(np.random.randn(5,4), index=dates, columns=
```

```
list('ABCD'))
    >>>print df
    ABCD
    2017-01-01 -1.008850  0.389870  0.422930 -0.239423
    2017-01-02 -1.804773 -0.122929 -0.570285  1.609145
    2017-01-03 -0.214131 -1.887011  0.546617 -1.250311
    2017-01-04  0.862515  0.417958 -0.506498 -0.330921
    2017-01-05  1.225338 -0.182110  1.117512 -1.338410
    df = pd.DataFrame({"a":1, "b":2, "c":3, "d":4, "e":("e1", "e2",
"e3")}) #传入 Python 的 dict 创建 DataFrame 数组
    >>>print df
    abcde
    0  1  2  3  4  e1
    1  1  2  3  4  e2
    2  1  2  3  4  e3
    >>>#================创建 Panel 对象================
    >>>p = pd.Panel(np.random.randn(2, 5, 4), items=['a', 'b'],
    ...           major_axis=pd.date_range('1/1/2016', periods=5),
    ...           minor_axis=['c', 'd', 'e', 'f'])
    >>>print p
    <class 'pandas.core.panel.Panel'>
    Dimensions: 2 (items) x 5 (major_axis) x 4 (minor_axis)
    Itemsaxis: atob
    Major_axis axis: 2016-01-01 00:00:00 to 2016-01-05 00:00:00
    Minor_axis axis: ctof
```

（3）数据查看

```
    >>>s = pd.Series([1, 2, 3, 4, 5])
    >>>df = pd.DataFrame(np.random.randn(5, 3), index=["a", "b", "c",
"d", "e"], columns=["item1", "item2", "item3"])
    >>>#================head 函数================
    ... s.head()
    0    1
    1    2
    2    3
    3    4
    4    5
    dtype: int64
    >>>df.head(3)
    item1     item2     item3
    a -0.954088  0.244918  1.145347
    b -0.807056  1.180588 -0.153845
    c  0.100668  0.162599 -0.320771
```

```
>>>#================tail 函数================
... s.tail(3)
2    3
3    4
4    5
dtype: int64
>>>df.tail(2)
item1     item2     item3
d  0.741110 -0.020918 -0.482747
e  0.756976 -0.112055  0.335975
#========通过 index、columns、values 查看 df 数据================
>>>df.index
Index([u'a', u'b', u'c', u'd', u'e'], dtype='object')
>>>df.columns
Index([u'item1', u'item2', u'item3'], dtype='object')
>>>df.values
array([[-0.95408841,  0.24491757,  1.14534653],
       [-0.80705606,  1.18058753, -0.15384471],
       [ 0.10066784,  0.16259905, -0.32077148],
       [ 0.74111005, -0.02091765, -0.48274747],
       [ 0.75697607, -0.11205458,  0.3359747 ]])
>>>#================df 行、列互换================
... df.T
abcde
item1 -0.954088 -0.807056  0.100668  0.741110  0.756976
item2  0.244918  1.180588  0.162599 -0.020918 -0.112055
item3  1.145347 -0.153845 -0.320771 -0.482747  0.335975
>>>#================df 排序查看================
... df.sort_index(axis=1, ascending=False)
item3     item2     item1
a  1.145347  0.244918 -0.954088
b -0.153845  1.180588 -0.807056
c -0.320771  0.162599  0.100668
d -0.482747 -0.020918  0.741110
e  0.335975 -0.112055  0.756976
```

（4）数据选择

```
>>> #内容：数据选择
>>>df = pd.DataFrame(np.random.randn(5, 3), index=["a", "b", "c",
"d", "e"], columns=["item1", "item2", "item3"])
>>>#获取列
... df["item1"]
a   -0.799240
```

```
b    1.354446
c    0.250335
d   -0.203544
e    0.703325
Name: item1, dtype: float64
>>>#获取行
... df[0:1]
item1    item2    item3
a -0.79924 -0.612019 -1.716188
>>>#通过索引获取数据
... df.loc["a"]
item1   -0.799240
item2   -0.612019
item3   -1.716188
Name: a, dtype: float64
>>>df.loc["a", ["item1", "item3"]]
item1   -0.799240
item3   -1.716188
Name: a, dtype: float64
>>>#通过位置获取数据
>>>df.iloc[2]
item1    0.250335
item2    0.067347
item3    0.058785
Name: c, dtype: float64
>>>df.iloc[2:3, [0, 1]]
item1    item2
c  0.250335  0.067347
>>>#获取指定的单个数据
... df.iat[1, 1]
0.086954440471 61624
```

（5）数据设置

```
>>>df = pd.DataFrame(np.random.randn(5, 3), index=["a", "b", "c", 
"d", "e"], columns=["item1", "item2", "item3"])
>>>print df
item1    item2    item3
a  0.155408  0.128694  1.385727
b -0.937322  0.789462  0.636947
c  0.203996 -0.573447 -0.136809
d  0.155883 -0.537128 -0.480155
e -0.626413  0.854594  0.448322
>>>#设置单个值
```

```
... df.iat[1,1] = 0.99999
>>>print df
item1     item2     item3
a  0.155408  0.128694  1.385727
b -0.937322  0.999990  0.636947
c  0.203996 -0.573447 -0.136809
d  0.155883 -0.537128 -0.480155
e -0.626413  0.854594  0.448322
>>>#设置一个序列
... df["item1"] = [0.1, 0.2, 0.3, 0.4, 0.5]
>>>print df
item1     item2     item3
a    0.1  0.128694  1.385727
b    0.2  0.999990  0.636947
c    0.3 -0.573447 -0.136809
d    0.4 -0.537128 -0.480155
e    0.5  0.854594  0.448322
```

（6）缺失值处理

```
>>>df = pd.DataFrame(np.random.randn(5, 3), index=["a", "b", "c", "d", "e"], columns=["item1", "item2", "item3"])
>>>df1 = df.reindex(index=["a", "b", "c", "d"], columns=list(df.columns) + ['item4'])
>>>print df1
item1     item2     item3  item4
a -1.366190  0.384771 -0.242857    NaN
b  0.498889 -0.415037  1.490942    NaN
c -1.154120 -0.323287  0.120690    NaN
d -0.174297 -1.495528 -1.517813    NaN
>>>df1.iat[0, 3] = 0.9999
>>>print df1
item1     item2     item3   item4
a -1.366190  0.384771 -0.242857  0.9999
b  0.498889 -0.415037  1.490942     NaN
c -1.154120 -0.323287  0.120690     NaN
d -0.174297 -1.495528 -1.517813     NaN
>>>#去除带有 Nan 空值的行
... df1.dropna(how='any')
item1     item2     item3   item4
a -1.36619  0.384771 -0.242857  0.9999
>>>#填补 Nan 空值
... df1.fillna(value = 0.1234)
item1     item2     item3   item4
```

```
a -1.366190  0.384771 -0.242857  0.9999
b  0.498889 -0.415037  1.490942  0.1234
c -1.154120 -0.323287  0.120690  0.1234
d -0.174297 -1.495528 -1.517813  0.1234
>>>#获取控制布尔值掩码
>>>pd.isnull(df1)
item1  item2  item3  item4
a False False False False
b False False False True
c False False False True
d False False False True
```

（7）汇总与统计

```
>>>df = pd.DataFrame({"a":[1, 2, 3], "b":[1, 3, 5], "c":[5, 4, 1], "d":[5, 6, 6], "e":[1, 7, 8]}, index=["r1", "r2", "r3"])
>>>print df
abcde
r1  1  1  5  5  1
r2  2  3  4  6  7
r3  3  5  1  6  8
#统计信息描述
>>>df.describe()
abcde
count  3.0  3  3.000000  3.000000  3.000000
mean   2.0  3  3.333333  5.666667  5.333333
std    1.0  2  2.081666  0.577350  3.785939
min    1.0  1  1.000000  5.000000  1.000000
25%    1.5  2  2.500000  5.500000  4.000000
50%    2.0  3  4.000000  6.000000  7.000000
75%    2.5  4  4.500000  6.000000  7.500000
max    3.0  5  5.000000  6.000000  8.000000
#获取单项指标，注意传入 0 或者 1 的差别
>>>df.count(0)
a    3
b    3
c    3
d    3
e    3
dtype: int64
>>>df.count(1)
r1    5
r2    5
r3    5
```

```
dtype: int64
>>> #内容：类 SQL 操作
>>>df = pd.DataFrame({"a":[1, 2, 3], "b":[1, 3, 5], "c":[5, 4, 1],
"d":[5, 6, 6], "e":[1, 7, 8]}, index=["r1", "r2", "r3"])
>>>print df
abcde
r1  1  1  5  5  1
r2  2  3  4  6  7
r3  3  5  1  6  8
>>>#类似 where 操作，其本质是先获取布尔掩码，之后获取值
... df[df.a> 1]
abcde
r2  2  3  4  6  7
r3  3  5  1  6  8
>>> #类似 insert 操作
>>>s = df.loc["r1"]
>>>print s
a    1
b    1
c    5
d    5
e    1
Name: r1, dtype: int64
>>>df.append(s, ignore_index = False)
abcde
r1  1  1  5  5  1
r2  2  3  4  6  7
r3  3  5  1  6  8
r1  1  1  5  5  1
>>>#类似 union 操作
... pieces = [df[:1], df[1:2], df[2:]]
>>>pd.concat(pieces)
abcde
r1  1  1  5  5  1
r2  2  3  4  6  7
r3  3  5  1  6  8
>>>#类似 join 操作
... left = pd.DataFrame({'key': ['foo', 'foo'], 'lval': [1, 2]})
>>>print left
keylval
0  foo    1
1  foo    2
>>>left = pd.DataFrame({'key': ['id1', 'id2'], 'lval': [1, 2]})
>>>right = pd.DataFrame({'key': ['id2', 'id3'], 'lval': [5, 6]})
>>>print left
```

```
keylval
0  id1    1
1  id2    2
>>>print right
keylval
0  id2    5
1  id3    6
>>>pd.merge(left, right, on='key')
keylval_xlval_y
0  id2      2      5
>>> #类似 groupby 操作
>>>s = pd.Series([1, 2, 3, 4, 5], index=["a", "b", "c", "d", "e"],
name="r1")
>>>df1 = df.append(s, ignore_index = False)
>>>print df1
abcde
r1  1  1  5  5  1
r2  2  3  4  6  7
r3  3  5  1  6  8
r1  1  2  3  4  5
>>>df1.groupby("a").sum()
bcde
a
1  3  8  9  6
2  3  4  6  7
3  5  1  6  8
```

（8）时间序列

时间序列基于 Timestamp 和 DatetimeIndex，以及 Period 和 PeriodIndex 进行操作。其主要用途是作为 Series 或者 DataFrame 的索引，并进行一系列的时间序列相关的操作。下面我们首先通过示例，对 Timestamp 和 Period 的数据形式做直观的展示，再接着说明如何生成以及应用该时间序列作为索引，示例如下。

```
>>>#创建 Timestamp
...pd.Timestamp('2017-01-01')
Timestamp('2017-01-01 00:00:00')
>>>pd.Timestamp(pd.datetime(2017, 1, 1))
Timestamp('2017-01-01 00:00:00')
>>>#创建 Period
... pd.Period('2017-01')
Period('2017-01', 'M')
```

```
>>>pd.Period('2017-01', freq = 'D')
Period('2017-01-01', 'D')
>>> #DatatimeIndex 索引生成的三种方式，示例如下：
>>>#方式一
... dates = [pd.Timestamp(pd.datetime(2012, 5, 1)), pd.Timestamp
(pd.datetime(2012, 5, 2)), pd.Timestamp(pd.datetime(2012, 5, 3))]
>>>index = pd.DatetimeIndex(dates)
>>>print index
DatetimeIndex(['2012-05-01',    '2012-05-02',    '2012-05-03'],
dtype='datetime64[ns]', freq=None, tz=None)
>>>#方式二
... index = pd.date_range('2017-1-1', periods=3, freq='D')
>>>print index
DatetimeIndex(['2017-01-01',    '2017-01-02',    '2017-01-03'],
dtype='datetime64[ns]', freq='D', tz=None)
>>>#方式三
... #date_range 生成的是日历时间，bdate_range 生成的是工作日时间
... index = pd.bdate_range('2017-1-1', periods=10)
>>>print index
DatetimeIndex(['2017-01-02',    '2017-01-03',    '2017-01-04',
'2017-01-05','2017-01-06',    '2017-01-09',    '2017-01-10',
'2017-01-11','2017-01-12',    '2017-01-13'],dtype='datetime64[ns]',
freq='B', tz=None)
>>> #PeriodIndex 索引生成的三种方式，示例如下：
>>>#方式一
... prng = pd.period_range('1/1/2017', '3/1/2017', freq='M')
>>>print prng
PeriodIndex(['2017-01',  '2017-02',  '2017-03'],  dtype='int64',
freq='M')
>>>#方式二
... pd.PeriodIndex(['2017-1', '2017-2', '2017-3'], freq='M')
PeriodIndex(['2017-01',  '2017-02',  '2017-03'],  dtype='int64',
freq='M')
>>>#方式三
... >>>pd.PeriodIndex(start='2017-01', freq='3M', periods=4)
>>>PeriodIndex(['2017-01',  '2017-04',  '2017-07',  '2017-10'],
dtype='period[3M]', freq='3M')
>>> #上述内容简要介绍了时间索引，接下来的示例将进一步说明时间索引作为 Series
和 DataFrame 的索引，以及 resample（重采样）函数的使用
>>>rng = pd.date_range('1/1/2017', periods=6, freq='M')
>>>ts.resample('2M', how="sum")
2017-01-31   384
2017-03-31   472
2017-05-31   974
2017-07-31   304
```

```
2017-09-30   492
2017-11-30   594
2018-01-31   418
Freq: 2M, dtype: int64
>>>rng = pd.date_range('1/1/2017', periods=6, freq='M')
>>>ts = pd.Series(np.random.randint(0, 500, len(rng)), index=rng)
>>>print ts
2017-01-31    91
2017-02-28   248
2017-03-31   256
2017-04-30   193
2017-05-31   482
2017-06-30   100
Freq: M, dtype: int64
>>>ts.resample('3M').sum()
2017-01-31    91
2017-04-30   697
2017-07-31   582
Freq: 3M, dtype: int64
>>>ts.resample('3M').mean()
2017-01-31    91.000000
2017-04-30   232.333333
2017-07-31   291.000000
Freq: 3M, dtype: float64
```

（9）文件 I/O

同上节介绍的 Numpy 一样，Pandas 提供了一系列简单易用的文件 I/O 函数，可以把 Series 以及 DataFrame 的数据以多种形式保存到文件，并从文件读取到内存中。可以保存的形式包括：csv、hdf、excel、json、html、sql、stata、sas、clipboard、pickle。此处简单介绍 csv，excel 的操作方式，其他格式的使用与此类似，可以参考官网进一步学习。

```
>>>df = pd.DataFrame(np.random.randn(4,3), index=["r1", "r2", "r3", "r4"], columns=list('abc'))
>>>#csv 文件的存取
>>>df.to_csv("df.csv")
>>>pd.read_csv("df.csv")
Unnamed: 0        abc
0        r1  0.242875  0.360095 -1.639387
1        r2  2.335254 -0.484403  0.776782
2        r3 -1.536690 -0.691591 -1.762023
```

```
3        r4 -0.423772  0.735941  0.206301
>>>excel 文件的存取
>>>df.to_excel('df.xlsx', sheet_name='Sheet1')
>>>pd.read_excel("df.xlsx", 'sheet1', index_col = None)
abc
0        r1  0.242875  0.360095 -1.639387
1        r2  2.335254 -0.484403  0.776782
2        r3 -1.536690 -0.691591 -1.762023
3        r4 -0.423772  0.735941  0.206301
```

2.2.3 Scipy

Scipy 是一款高性能数学计算函数库，它包含科学计算、工程计算等领域的常用数学方法实现。与 Pandas 相比，两者都是基于 Numpy 构建，但 Pandas 偏重数据的统计，而 Scipy 则着重实现数学计算。Scipy 类似 MATLAB 的 toolbox，或者 GSL (GNU Scientific Library for CandC++)。学懂 Scipy 实现的函数库需要一定的高等数学、数据挖掘以及信号处理等理论基础，读者可通过后续的章节或者参考文献学习。

Scipy 根据不同的功能应用划分成不同的子模块，子模块之间基本是相互独立的。表 2-1 对各模块及其功能做了简要的描述。

官网手册

https://docs.scipy.org/doc/scipy/reference/

内容概览

表 2-3　Scipy 内容概览

模　　块	说　　明
scipy.cluster	矢量量化、*k*-means、层次聚类
scipy.constants	物理和数学常数
scipy.fftpack	傅里叶变换
scipy.integrate	积分程序
scipy.interpolate	插值
scipy.io	数据的输入输出
scipy.linalg	线性代数程序

续表

模　　块	说　　明
scipy.ndimage	*N* 维图像包
scipy.odr	正交距离回归
scipy.optimize	优化
scipy.signal	信号处理
scipy.sparse	稀疏矩阵
scipy.spatial	空间数据结构和算法
scipy.special	任何特殊数学函数
scipy.stats	统计

表 2-3 清晰地说明了各模块的内容。由于 Scipy 的 API 接口使用较为简单，而它设计上的数学理论又并非本书要着重讲解的内容，因此，此处不再赘述。

2.2.4　Matplotlib

Matplotlib 是绘制高质量 2D 图像的 Python 库。可以方便地绘制折线图、柱状图、散点图等多种图形；丰富的样式和灵活的标注可以更加直观地呈现图像；此外，Matplotlib 的 API 的设计简洁清晰，官网的文档详尽易懂。这些特点使它成为 Python 实现数据可视化的一把利器。

官网手册

http://matplotlib.org/contents.html

内容概述

表 2-4　两种类型的 API 使用

API 类型	说　　明
面向过程 API	Matplotlib 提供一套仿照 MATLAB 的接口调用，逐条地调用函数命令来绘制图像。以画一个平面折线来做比喻：第一步先拿出一张纸；第二步选定位置画上直角坐标系；第三步画上折线；第四步标上刻度
面向对象 API	与面向过程 API 相比，它把图像的各个部分看成对象，由对象自己负责自己的操作。仍然以画一个平面折线做比喻：纸张是一个对象，负责设定背景，大小等。直角坐标系是一个对象。X 轴、Y 轴分别是一个对象 画图的过程如下，纸张对象设置好自己的大小、背景之后返回一个坐标对象；坐标

续表

API 类型	说　　明
	对象根据数据画出折线；同时，返回其包含的 X 轴以及 Y 轴对象。依次类推，最终完成一幅图片的绘制

示例版本

版本号：1.53

日期：2016.12.05

示例讲解

用两种类型的 API 可以做出如图 2.1 所示的图像，请注意代码中尽量使用详尽的注释。读者可依据代码体会两种绘图思想。更多的细节处理可参考官网去学习与实现。

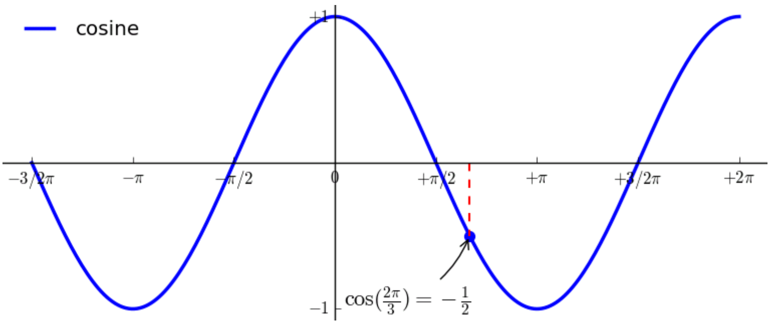


图 2.1　Matplotlib 绘制 cosine 图像

面向过程作图

面向过程作图是指利用 pyplot 包提供的函数命令来逐步绘图。步骤如下：先创建 figure，类似于图纸；再调用函数创建坐标系，绘制图像；最后调整刻度及标签、添加注释并依次绘制图像的各个细节。

pyplot 的绘图命令只是对面向对象做图接口的封装，实现类似 MATLAB 的作图方式。它的代码示例如下：

```
    import numpy as np
    import matplotlib.pyplot as plt
    #X 轴、Y 轴数据
    X = np.linspace(-1.5 * np.pi, 2 * np.pi, 256,endpoint=True)
    C= np.cos(X)
    #plt 创建图纸
    plt.figure(figsize=(10, 4), dpi=100)
    #plt 定位坐标
    ax = plt.subplot(111)
    #调整坐标轴
    ax.spines['right'].set_color('none')
    ax.spines['top'].set_color('none')
    ax.xaxis.set_ticks_position('bottom')
    ax.spines['bottom'].set_position(('data',0))
    ax.yaxis.set_ticks_position('left')
    ax.spines['left'].set_position(('data',0))
    #plt 画图
    plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-", label=
"cosine")
    #axis 对象设置数据轴宽度、修改刻度、刻度标签
    plt.xlim(X.min()*1.1, X.max()*1.1)
    plt.xticks([-1.5 * np.pi, -np.pi, -np.pi/2, 0, np.pi/2, np.pi, 1.5
* np.pi, 2*np.pi],
             [r'$-3/2\pi$',    r'$-\pi$',    r'$-\pi/2$',    r'$0$',
r'$+\pi/2$', r'$+\pi$',r'$+3/2\pi$',r'$+2\pi$'])
    plt.ylim(C.min()*1.1,C.max()*1.1)
    plt.yticks([-1, +1],
             [r'$-1$', r'$+1$'])
    #plt 绘制注释
    t = 2*np.pi/3
    plt.plot([t,t],[0,np.cos(t)],color ='red', linewidth=1.5, linestyle
="--")
    plt.scatter([t,],[np.cos(t),], 50, color ='blue')
    plt.annotate(r'$\cos(\frac{2\pi}{3})=-\frac{1}{2}$',
    xy=(t, np.cos(t)),  xycoords='data',
    xytext=(-90, -50), textcoords='offsetpoints', fontsize=16,
    arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))
    #plt 调整图例
    plt.legend(loc='upperleft', frameon=False)
    #plt 展示图像
    plt.show()
```

面向对象作图

Matplotlib 提出了 Object Container（对象容器）的概念，它有 Figure、Axes、Axis、Tick 四种类型的对象容器。Figure 负责图像大小、位置等操作；Axes 负责坐标系位置、绘图等操作；Axis 负责坐标轴的操作；Tick 负责刻度的相关操作。

四种对象容器之间是层层包含的关系。Figure 包含 Axes、Axes 包含 Axis、Axis 包含 Tick。我们用一种不太确切但是形象的方式来说明“包含”的含义以及作图的过程：Figure 定位图像大小、位置之后返回一个 Axes 对象；Axes 对象绘制图片、调整坐标轴位置之后发回一个 Axis 对象；Axis 对象又可以来设置 *X*、*Y* 坐标的显示长度，刻度以及刻度标签等，之后返回 Tick 对象；Tick 对象则可以格式化各个刻度的样式等。代码示例如下。

```
    import numpy as np
    import matplotlib.py plot as plt
    #X 轴、Y 轴数据
    X = np.linspace(-1.5 * np.pi, 2 * np.pi, 256,endpoint=True)
    C = np.cos(X)
    #创建 figure 对象
    fig = plt.figure(figsize=(10,4), dpi=100, tight_layout=True)
    #figure 对象获取 axes 对象
    ax  = fig.add_subplot(111)
    #axes 对象绘图
    ax.plot(X,   C,   color="blue",   linewidth=2.5,   linestyle="-",
label="cosine")
    #axes 对象调整坐标轴
    ax.spines['right'].set_color('none')
    ax.spines['top'].set_color('none')
    ax.spines['bottom'].set_position(('data',0))
    ax.spines['left'].set_position(('data',0))
    #axes 对象获取 axis 对象
    xAxis = ax.get_xaxis()
    yAxis = ax.get_yaxis()
    #axis 对象设置数据轴宽度、修改刻度、刻度标签
    xAxis.set_data_interval(X.min()*1.1, X.max()*1.1)
    xAxis.set_ticks([-1.5 * np.pi, -np.pi, -np.pi/2, 0, np.pi/2, np.pi,
1.5 * np.pi, 2*np.pi])
    xAxis.set_ticklabels([r'$-3/2\pi$', r'$-\pi$', r'$-\pi/2$', r'$0$',
r'$+\pi/2$', r'$+\pi$',r'$+3/2\pi$',r'$+2\pi$'])
    yAxis.set_data_interval(C.min()*1.1, C.max()*1.1)
```

```
    yAxis.set_ticks([-1, +1])
    yAxis.set_ticklabels([r'$-1$', r'$+1$'])
    #axis 对象设置刻度位置
    xAxis.set_ticks_position('bottom')
    yAxis.set_ticks_position('left')
    #figure 添加 ax2 对象，与 ax 共用一个区域
    ax2  = fig.add_subplot(111)
    #axes 对象作图，以及添加注释
    t = 2*np.pi/3
    ax2.plot([t,t],[0,np.cos(t)],color ='red', linewidth=1.5, linestyle=
"--")
    ax2.scatter([t,],[np.cos(t),], 50, color ='blue')
    ax2.annotate(r'$\cos(\frac{2\pi}{3})=-\frac{1}{2}$',
    xy=(t, np.cos(t)),  xycoords='data',
    xytext=(-90, -50), textcoords='offsetpoints', fontsize=16,
    arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))
    #axes 对象添加图例
    ax.legend(loc='upperleft', frameon=False)
    #展示图像
    fig.show()
```

3 基础分析

概率论与统计学是基础分析、数据挖掘、深度学习的理论基石，包含了概率、随机变量分布、大数定律、参数估计、假设检验等多个层面的知识。本章以“统计量”和“概率分布”为切入点，阐述在特定场景下，如何对数据做有理有据的分析及解释。

3.1 场景分析与建模策略

针对不同的业务场景，数据工程师需要采用不同的建模策略，并且在大多数的场景中，简单直观的基础分析策略要比复杂的挖掘算法更加高效。“统计量”和“概率分布”是日常分析工作中适用范围最广、使用频次最高的两种基础分析方法，也是后续数据挖掘以及深度学习模型的理论基础。虽然这两种分析方法简单易懂，但是往往很难被正确的应用。

3.1.1 统计量

“统计量”或“抽样统计量”是样本测量的一种属性，统计学定义一个统计量

为一个总体参数的点估计量。常用的统计量包括“均值”“方差”“中位数”等，反映了数据集合所呈现的不同统计特性。本节针对这些统计特征，详细介绍各统计量在不同场景中的应用。

1．均值

（1）场景描述

每个班级数学成绩的优劣对比，多个车间工人效率的高低比较，类似这些整体均衡的比较，都可以用均值度量。

但是，均值对离群值十分敏感。只有在数据分布相对均匀的情况下，才能很好地反映问题。对于存在离群值的数据集，中位数是一种更好的选择。我们先给出均值的定义，并在下面的示例中对相关问题做详细的说明。

（2）均值

给定一个具有 n 个样本的集合，$X = \{x_1, x_2, x_3, \cdots, x_n\}$　$n \geqslant 1$; 它的均值（mean）计算公式如下：

$$\text{mean}(x) = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{3.1}$$

（3）示例说明

例 3.1 考虑工厂有 A、B 两个车间，每个车间有若干工人。下班之前，监工需要统计哪个车间的工人加工零件的效率更高。A、B 车间每个工人当天加工零件数分别为

① A = {100，120， 98，101，90};

mean(A) = (100 + 120 + 98 + 101 + 90) / 5 = 101.80

② B= {99, 98, 110, 102, 114, 99, 102, 101};

mean(B) = (99 + 98 + 110 + 102 + 114 + 99 +102 + 101) / 8 = 103.13

因此，由均值可知，B 车间工人工作效率高于 A 车间。

但假如 A、B 的数据是像下面这样的，那么均值并不能很好地度量效率。

① A = {120，125，125，120，**30**};

mean(A)　= (120 + 125 + 125 + 120 + 10) / 5 = 100.00

② B={99, 98, 110, 102, 114, 99, 102, 101};

mean(B)　= (99 + 98 + 110 + 102 + 114 + 99 +102 + 101) / 8 = 103.13

对比均值可知，B 车间工人效率较高，然而从直观数据看，车间 A 的工人效率应该是整体高于车间 B 的。不难发现，导致均值失去度量效果的原因是 A 中存在离群值“30”。

严格来说，只有数据集合服从均匀分布时，均值才能较为准确地反映数据的均衡程度。通常情况下，先筛选出明显的离群值，再进行均值的度量；或者用下面所述的“中位数”来近似地说明均衡程度的问题。

2．中位数

（1）场景描述

如例 3.1 所述，当数据集中存在较多的离群点，均值无法合理地反映均衡度量的时候，中位数往往能更好地反映出数据集的中间程度。

（2）中位数

给定一个具有 n 个样本的集合，X = {x_1 , x_2, x_3，⋯ , x_n}，$x_n > x_{n-1}$，$n \geqslant 1$；中位数（median）的计算公式如下：

$$\text{median}(x) = \begin{cases} x_{\frac{n+1}{2}} & \text{如果}n\text{是奇数} \\ (x_{\frac{n}{2}} + x_{\frac{n+1}{2}})/2 & \text{如果}n\text{是偶数} \end{cases} \tag{3.2}$$

（3）示例说明

根据例 3.1 中第二组假设数据，对比中位数和均值在度量工作效率时的效用。

① A = {120，125，125，120，**30**};

② B={99, 98, 110, 102, 114, 99, 102, 101};

由统计数据及表 3-1 可以看出，当存在离群值时，中位数更好地说明了问题。因此，灵活运用均值和中位数，能更好地度量均衡程度。

表 3-1　车间 A、B 均值、中位数

车　　间	均　　值	中位数
A	100.00	125
B	103.13	108

3．众数

（1）场景描述

均值、中位数只是从一定程度反映出数据集的平均程度，难以说明哪些数据具有代表性。比如，哪款商品最热销，哪篇文章最受欢迎等，此时，应考虑采用众数作为统计量，能更好地说明问题。

（2）众数

众数是一组数据中出现次数最多的数值。

（3）示例说明

例 3.2 假设商场新进 A、B、C、D 四种类型的加湿器，销售 3 个月之后，想知道哪款加湿器更受欢迎，通过统计每款加湿器的销售量得到如表 3-2 的数据。

表 3-2　A、B、C、D 四种类型加湿器 3 个月的销售量

加湿器	A	B	C	D
销售量	782	343	898	408

由表 3-2 可知，C 出现了 898 次，是众数，因此，可以说明 C 类型的加湿器

更受欢迎。

众数不受个别异常值的影响，可在数据缺失较多时，粗略地求出具有代表性的数值，可以和均值、中位数配合使用，合理说明数据集中那些更具代表性的数据。

4．方差

（1）场景描述

当两名学生多次模拟考试的数学成绩均值相同时，如何判断谁的成绩更稳定呢？当两地某月每天的平均气温相同，如何判断哪一地温度按天波动较大呢？类似这样的问题，可以用方差统计量做出合理的解释。

（2）方差

给定一个具有 n 个样本的集合，$X=\{x_1, x_2, x_3, \cdots, x_n\}$　$n\geqslant 1$；方差（variance）的计算公式如下：

$$\text{variance}(x)=\frac{\sum_{i=1}^{n}(x_i-\bar{x})^2}{n-1} \tag{3.3}$$

（3）示例说明

例 3.3 假设某学校高三年级本学期共进行了 10 次模拟考试。其中，小明和小兴在历次模拟考试中的成绩如表 3-3 所示。

表 3-3　历次数学模拟考试成绩

序号	1	2	3	4	5	6	7	8	9	10
小明	95	99	98	90	92	92	99	91	90	90
小兴	99	99	91	92	91	99	92	91	90	92

依据表 3-3 的数据，小明和小兴的均值为 93.6。因此，两位同学在数学上的平均能力旗鼓相当。那么，方差的统计可以进一步说明谁更稳定、更优秀。通过计算得出方差分别为 13.04 和 12.84，说明小兴的成绩相对小明波动较小。因此，

小兴的表现更优秀。

5．协方差与相关系数

（1）场景描述

方差反映了一组数据自身的波动情况，而协方差则反映了两组数据之间的线性相关性。比如例 3.3 可以转换成：如何从整体趋势上说明小明的成绩是随着小兴成绩的提高而提高（可能是考试难易程度的变化导致趋势相同）的，还是刚好相反？比如昼夜温差的变化和水果的含糖量是否具有一致性？这类问题一定程度上皆可以通过协方差度量并说明。

（2）协方差

给定两个具有 n 个样本的集合，$X=\{x_1, x_2, x_3, \cdots, x_n\}$，$Y=\{y_1, y_2, y_3, \cdots, y_n\}$，$n\geqslant 1$;协方差（covariance）的计算公式如下：

$$\mathrm{cov}(X,Y)=\frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{n-1} \tag{3.4}$$

（3）示例说明

仍以例 3.3 的数据为例，小明和小兴成绩的协方差为 4.822。协方差为正，说明小明和小兴的成绩正相关。导致这种趋势的原因可能是每次考试的难易程度：试题较难时，两人考试成绩都较低；试题较容易时，两人考试成绩都高。

同时，应该注意协方差的正负只是反映线性相关的方向，并不能反映线性相关的程度。有兴趣的读者可以研究一下相关系数，相关系数的取值在[-1,1]之间，其在协方差的基础上进一步表明了数据的相关程度。

3.1.2 概率分布

概率分布是概率论的一个概念，可以狭义地认为概率分布是指随机变量所符合的概率分布函数。符合某种场景的随机变量，都会遵循一定的概率分布，比如

“单位时间内某随机事件发生的次数”往往符合泊松分布，“精密元器件的误差”一般符合正态分布等。本节结合应用场景，逐步讲解概率分布在数据分析中的应用。

1．泊松分布

（1）场景描述

泊松分布适合描述单位时间内随机事件发生次数的概率分布。这个单位时间可以是“每分钟”“每小时”等等。比如小贩每天卖出的豆浆碗数是 1、2、3 碗……的概率分别是多少，这个分析可以帮助他决定每天该准备多少碗豆浆；比如商场每天卖出去的某款皮鞋的数量是 1、2、3 双……的概率是多少，并帮助商场决定进多少货物，最大程度地利用仓储空间。

对于满足上述场景的随机事件，通常情况下是满足泊松分布的；同时，对于较为严苛的场景，可以通过卡方检验，有效说明该随机事件是否符合泊松分布，具体卡方检验的方法此处不再赘述。

（2）泊松分布

参数 k 代表事件 X 发生的次数；参数λ是单位时间（或单位面积）内随机事件发生的平均次数。

$$P(x=k)=\frac{\lambda^k}{k!}\mathrm{e}^{-\lambda}, k=0,1,\cdots \tag{3.5}$$

（3）示例说明

例 3.4 假设有一早餐摊位，老板思考如何不让豆浆缺货，同时又尽可能没有剩余。他连续记录了 15 天之内每天的销售量，依次为：52, 55, 49, 43, 61, 53, 48, 40, 51, 53, 50, 42, 58, 51, 45。由上述场景可知，该数据序列属于泊松分布，λ=50。代入公式求得如图 3.1 所示的概率分布。

图 3.1 直观展示了豆浆每天售卖碗数的概率分布。该分布属于离散随机变量分布，X 轴为每天所卖豆浆的杯数，Y 轴为其概率值。表 3-4 列举了详细的数值数

据和对应的累计概率。

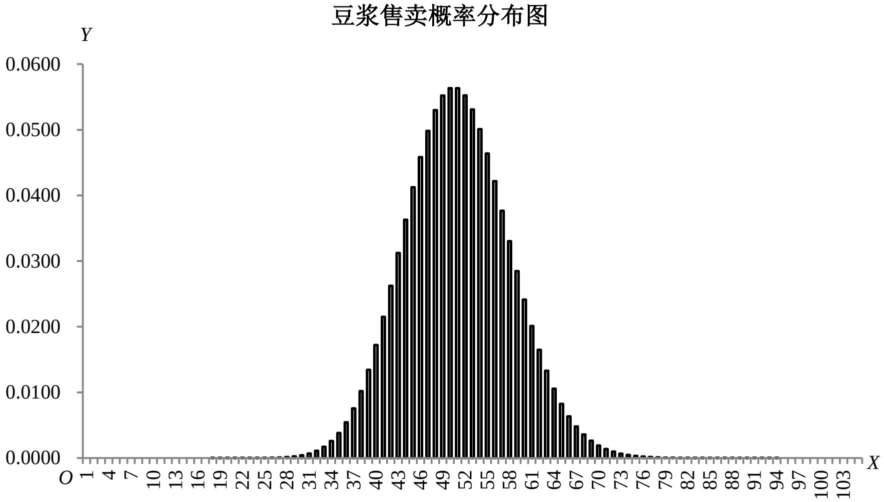


图 3.1　豆浆售卖概率分布图

表 3-4　依据泊松分布计算的豆浆日销量的概率

个　　数	概　　率	累计概率
0	0.0000	0.0000
1	0.0000	0.0000
...		
50	0.0563	0.5375
51	0.0552	0.5927
52	0.0531	0.6458
53	0.0501	0.6959
54	0.0464	0.7423
...		
70	0.0019	0.9970
71	0.0014	0.9980
72	0.0010	0.9987

由累计概率可知,如果每天预备 70 杯豆浆,那么 99.7%的情况下是不缺货的;同时，也说明了准备更多的豆浆对降低缺货率并没有大的作用，而且容易造成浪费。早餐摊位老板可以依据该概率统计的结果适当预备豆浆。

2．指数分布

（1）场景描述

泊松分布可以描述单位时间内独立事件发生的次数。而独立事件发生的时间间隔的概率，则可以使用指数分布描述。比如公交站台每分钟平均有 3 个人等公交，那么第一个人到公交站台之后，第二个人到来的时间间隔分别为 0.5 分钟、1 分钟、2 分钟……的概率是多少。该类问题可以使用指数分布做合理解释。

同时，应该注意到在泊松分布中，随机变量 X 代表事件发生的次数，1 次、2 次等依此类推，其变量是离散值，属于离散随机变量的分布；指数分布研究的随机变量是时间，比如 0.5 分钟、1 分钟、1.2 分钟等依此类推，其属于连续随机变量的分布。

（2）指数分布

指数分布的概率密度函数计算公式如下:

$$f(x;\lambda)=\begin{cases}\lambda \mathrm{e}^{-\lambda x}, & x>0\\ 0, & x\leqslant 0\end{cases} \tag{3.6}$$

其中λ为单位时间内随机事件发生的次数。

（3）示例说明

例 3.5 假设某客服中心平均每分钟接通 1 个咨询电话，为了合理配置客服人数，需要统计每次电话之间的时间间隔。

该场景属于指数分布的应用场景，因此，依据平均每分钟接通电话的个数，根据计算公式(3.6)求概率密度函数的分布如图 3.2 所示。

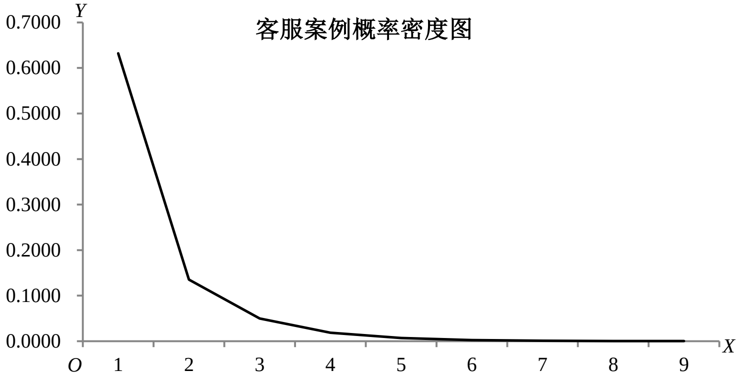


图 3.2　来电时延概率分布图

如图 3.2 所示，X 轴为“下次”电话响起的时间间隔，Y 轴是它对应的概率密度，曲线下方的面积代表概率值。图 3.2 可以大致说明在第一次接通电话之后的 0~2 分钟内，电话再次响起的概率很高（0~2 分钟内曲线下方的面积占比较大，说明概率较高）。因此，结合单次通话的平均时长，可以进一步合理配置客服人员的数量，优化资源配置。

3．正态分布

（1）场景描述

和指数分布相同，正态分布研究的也是连续的随机变量。自然界和人类社会中大量的随机变量服从正态分布。比如，一片森林中每棵树木的高度，生产药物在剂量上的误差等。同时也要注意，在做出正态分布的假设之前，出于严谨的考虑，依然需要通过概率密度图或者相关检验来进一步说明要分析的数据集是否确实具有正态分布的特性。

（2）正态分布

正态分布的概率密度函数计算公式如下。

$$f(x;\mu;\delta)=\frac{1}{\delta\sqrt{2\pi}}\mathrm{e}^{-\frac{(x-\mu)^2}{2\delta^2}} \tag{3.7}$$

其中 μ 代表随机变量的均值，δ 代表随机变量的方差。

（3）示例说明

例 3.6 假设某精密器件需要定期更换，已知该器件的正常使用时长属于正态分布。现在维修工人想知道多少天检修一次比较合理。通过一定时期的统计得出，该器件的平均正常使用天数 μ=20，方差 δ 为 1。由公式（3.7）可以求得该正态分布的概率密度，如图 3.3 所示。

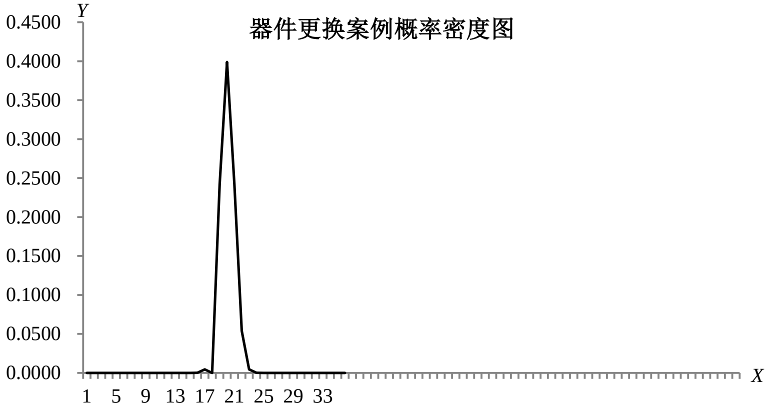


图 3.3　器件寿命的正态分布图

图 3.3 是该类器件正常使用天数的概率密度分布图。其中 *X* 代表时间，*Y* 代表其对应的概率密度。根据概率密度函数，可以求解得出正常使用天数在 18~22 天的概率是 95.94%，因此可以在 18 天之后对器件做一次彻底的检修，以达到事半功倍的效果。

4．Zipf 分布

（1）场景描述

正态分布和幂律分布是自然界多样繁杂的事件中普遍存在的两种分布形式。比如，人类的智力服从正态分布，而人类的财富则服从幂律分布。其中 Zipf（齐普夫）分布和 Pareto（帕累托）是幂律分布在离散随机变量和连续随机变量上的

两种具体形式。通常符合这两种分布的随机变量会呈现出“二八原则”和“长尾现象”。哈佛大学语言专家 Zipf 1935 年研究英文单词时发现，如果按单词出现的频率由高到低排名，其序号 1,2,3,⋯,n，与对应的频率 x_1, x_2 ,x_3,⋯,x_n，这两个序列呈简单的反比，这种关系称为 Zipf 定律。可用公式(3.8)量化计算。

（2）Zipf 分布

Zipf 分布的概率函数计算公式如下：

$$Y(X) = aX^{-b} \tag{3.8}$$

前面介绍的其他概率分布的计算公式一般与期望、或者方差紧密相关，而幂律分布的概率密度函数则不一样。公式(3.8)中的 X 为单词出现的频率排名名次，Y 为单词出现的频率，a、b 都为常数。可以依据已有数据，通过最小二乘法[13]等拟合算法来求解参数，本节不赘述拟合的具体过程，相关库函数已经实现，完全可以当成黑盒来使用。

（3）示例说明

例 3.7 假设某大型跨国公司年终对各销售人员的销售额占比排名，得出表 3-5 中的数据。

表 3-5　销售额占比排名

排　　名	销售额占比
1	10.00%
2	5.91%
3	4.34%
...	...
97	0.31%
98	0.31%
...	
177	0.19%
...	

依据数据绘制图 3.4 如下。

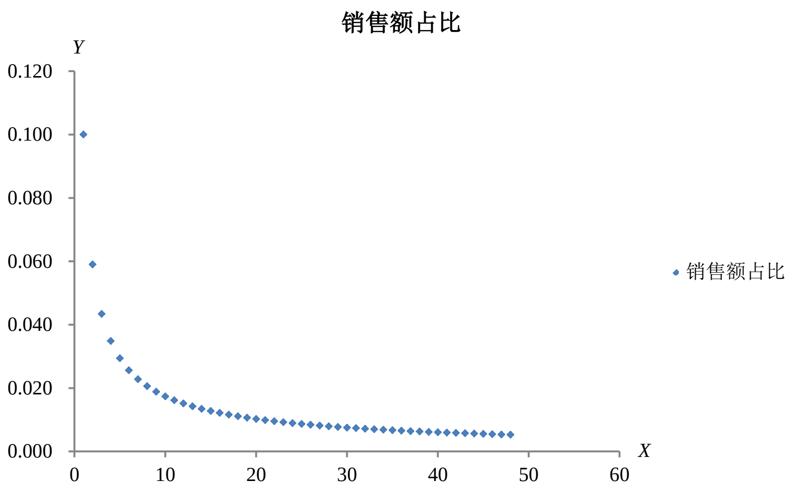


图 3.4 销售额占比排名

图 3.4 中的 *X* 轴代表销售额占比排名的次序，*Y* 轴代表销售额占比。销售额占比可以看成是 Zipf 试验中单词出现的频率。依据 Zipf 概率函数，可以求出排名第 *N* 位的销售人员的销售额占比。可以看出，少部分的销售人员完成了大多数的销售份额，这也正是“二八原则”的体现。能力较强的销售人员毕竟是少数，而这些少数的销售人员却完成了大部分的业绩。

3.2 实例讲解

3.2.1 谁的成绩更优秀

某体育队挑选运动员参加高级别百米田径比赛。运动员 A 和运动员 B 都符合参赛条件，但是只有一个名额。应该选择哪一位候选人更合适呢？

提问者抛出的问题往往是模糊的，且看起来与数据毫无关系。数据工程师的工作正是针对这些现实问题，寻求数据的支撑与解答。问题中“更合适的候选人”的潜在含义就是能力突出且发挥稳定的人，而均值和方差正是这两项特征的体现。

至此，我们初步明确了分析目标，即统计两名运动员过去比赛中百米时长的均值与方差，优先选取均值较小的运动员；在均值相等的情况下，选取方差较小的运动员。

继而我们需要搜集相关的比赛数据。假设过去一年运动员的赛事成绩如表 3-6 所示。

表 3-6　运动员过去一年的赛事成绩

运动员 A	11.23	11.56	13.05	12.46	12.18	13.10	13.33
运动员 B	13.02	13.12	13.54	10.49	12.30	12.22	12.22

Python 的 Numpy 库中的 ndarray 数组提供了 mean 和 var 方法,可以用来统计数据的均值与方差。现实中待分析的数据一般通过 I/O 操作函数，从文件或者数据库中导入。本例较为简单，代码中直接引用了表格的数据。

```
    import numpy as np
    import matplotlib.py plot as plt
    #运动员 A 比赛数据
    a = np.array([11.73, 11.56, 12.55, 12.46, 12.18, 13.10, 13.33], dtype
= np.float64)
    #运动员 B 比赛数据
    b = np.array([12.02, 12.12, 12.74, 12.79, 12.80, 12.22, 12.22], dtype
= np.float64)

    #对比均值
    print "对比 A 与 B 的均值："
    print "\t\tA：%s"%(round(a.mean(), 2))
    print "\t\tB：%s"%(round(b.mean(), 2))

    #对比方差
    print "对比 A 与 B 的方差："
    print "\t\tA：%s"%(round(a.var(), 2))
    print "\t\tB：%s"%(round(b.var(), 2))
```

程序运行结果如下。

```
对比 A 与 B 的均值：
                A：12.14
                B：12.14
对比 A 与 B 的方差：
```

```
A：0.69
B：0.41
```

我们需要审查代码或者用其他方式重新计算，以确保数据的准确性。最后根据数据分析得出结论：两位候选人成绩均值均为 12.14，在实力上可谓旗鼓相当；但是，由于运动员 B 的方差小于运动员 A，所以运动员 B 发挥相对稳定。因此，运动员 B 是本次田径运动的最佳人选。

3.2.2　应该库存多少水果

某大型水果超市每天从农产品市场批发水果。需要预估苹果每天的售卖数量，以制定合理的采购数量。该如何完成这件事情？

数据工程师要从实际的场景出发，合理地理解诉求，并依据数据制定解决方案。“制定合理的采购数量”的潜在含义是不能缺货，同时不能剩余太多，以保证水果的新鲜。那么如何既做到不缺货又不剩余呢？我们从数据的角度分析：如果能知道每天卖出水果的概率分布，那么就可以量化指标。比如，可以求出至多卖掉 100 个苹果的概率是 89%。因此可以尝试从概率分布的方向来切入。

前面已介绍，单位时间内事件发生的次数遵循泊松分布。因此，如果能够从历史售卖记录中，求出泊松分布的概率公式，就可以预估每天售卖的苹果个数。

至此，分析的目标已经逐渐清晰——基于历史售卖数据，求解苹果销售的泊松分布参数，进而求出每天售卖苹果个数的概率。需要特别说明的是，出于严谨的考虑，可以利用卡方检验来说明事件是否符合泊松分布。本书遵循一般情况，即该类事件都服从泊松分布。

在确定分析目标以及分析方案的基础上，采用 Python 作为数据处理的工具。Python 的 scipy.stats 包中，已经实现了“泊松分布”“正态分布”以及“指数分布”等多种分布的接口函数，这些接口函数可以直接应用到数据处理中。如下述程序所示，首先模拟训练数据集；其次，求解该模拟数据集的泊松分布的 λ 参数，得出概率公式；继而，依据该公式求解每天售卖苹果个数的累计概率，即每天卖出

的苹果数量小于等于 N 的概率。最后，以图形化的方式（图 3.5）展示分析结果，并给出最终结论。

```
    import numpy as np
    from scipy import stats
    import matplotlib.pyplot as plt

    # 假设店主记录了 90 天售卖数据，模拟数据如下
    mean = 467
    days = 90
    appleSellRecord = stats.poisson.rvs(mean, size=days)

    # 泊松分布的参数 λ 如下
    lambda = appleSellRecord.mean();

    # 由此计算每天卖出 300 个、301 个、302 个... 1000 个苹果的概率
    n=np.arange(0, 1000, 1)
    cumPro = stats.poisson.cdf(n, lambda)

    # 查看累计概率数值
    precision = lambdax: round(x, 4)
    cd = map(precision, cumPro)
    # 绘制累计概率分布图
    #X 轴、Y 轴数据
    X = n
    Y = cumPro

    #创建 figure 对象
    fig = plt.figure(figsize=(10,4), dpi=100, tight_layout=True)

    #figure 对象获取 axes 对象
    ax  =fig.add_subplot(111)

    #axes 对象绘图
    ax.plot(X, Y, color="blue", linewidth=2.5, linestyle="-", label=
"CumulativeProbability")

    #axes 对象调整坐标轴
    ax.spines['right'].set_color('none')
    ax.spines['top'].set_color('none')
    ax.spines['bottom'].set_position(('data',0))
    ax.spines['left'].set_position(('data',0))

    #axes 对象获取 axis 对象
```

```
xAxis = ax.get_xaxis()
yAxis = ax.get_yaxis()
xAxis.set_ticks_position('bottom')
yAxis.set_ticks_position('left')
xAxis.set_data_interval(-100, 1000)
xAxis.set_ticks(np.arange(-200, 1000, 100))
yAxis.set_data_interval(-1, 3)
yAxis.set_ticks(np.arange(-0.5, 2, 0.5))

#axes 对象添加图例
ax.legend(loc='uppercenter', frameon=False)

#展示图像
fig.show()
```

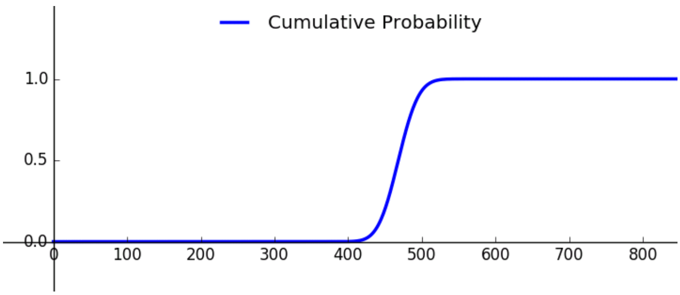


图 3.5　每天售卖苹果的累计概率密度图

图 3.5 展示了分析的结果。从中可知，400 之前的累计概率无限接近 0，且 500 以后的累计概率无限接近 1。说明每天售卖的苹果个数在 400 到 500 之间；同时，500 以后累计概率趋于平稳，说明每天购进 500 个左右苹果，能够保证不会缺货且不会剩余，而购进更多的苹果可能滞销。因此结论是水果超市应该每天购进 500 个苹果较为合理。

4 数据挖掘

与基础分析相比，数据挖掘汲取信息论、优化理论、搜索算法等多个领域的知识，形成了一套完善的数据分析策略。依据挖掘算法的目的，一般可将它分为四大类：分类、聚类、关联规则、回归。

市面上已有大量关于数据挖掘的经典书籍。本章没有采取“大而全”的方式详述所有挖掘算法,而是从实战的角度出发,讲述在实践中使用频率较高的算法，并重点阐述其中需要了解的关键知识点。

4.1 场景分析与建模策略

不同的挖掘建模策略具有不同的适用场景。“分类”是针对具有类别标签的训练数据集构建模型，并把该模型用于后续的样本分类；“聚类”则是针对无标签的训练数据集构建模型,根据样本之间的“距离”,让同类数据自动聚合的建模策略；“关联规则”通过统计数据出现的频次，定义事物之间相关性强弱；“回归”利用历史数据或者相关数据，实现合理的预测。本节逐次对各模型的建模理念、构建过程、具体案例进行详细介绍。

4.1.1　分类

分类，指通过大量的训练数据集建模，得出可以用于分类的数学公式或者概率模型，且该模型可以运用到后续的分类场景中。分类的训练集，一般都要先通过人工或者其他的方式标注类别，才能用于模型的训练。这种需要事先标注训练集类别才能建模的机器学习方式被称为“监督学习”。其中较为经典的算法包含决策树、朴素贝叶斯、支持向量机和最邻近分类器等。

1. 决策树分类

（1）模型的构建

我们通过示例说明决策树模型的构建过程。假设有如下场景：“银行征信系统”有客户信息的相关记录，如何使用这些数据构建信贷模型，以此来评估后续申请银行借贷的客户，从而有效地避免坏账。

第一步，我们需要从已有的用户信息中抽取相关数据，如表 4-1 所示。那么为什么会抽取这些特征项，是否有一定的方法或者依据？这些特征项的选取，并不存在既定有效的方法与理论，通常依据人为经验和对业务的理解，选择最有可能影响类别判断的特征值，并根据对后续建模效果的评估，不断调整到更优。

表 4-1　银行借贷系统训练数据集

用户 ID	是否有房	是否有车	是否已婚	本科及以上	拖　欠
1	是	否	是	是	否
2	否	是	否	否	是
3	是	是	否	否	否
4	是	否	是	否	否
5	否	否	是	是	是
6	是	是	是	是	否
7	否	是	否	是	否
…	…	…	…	…	…
N	是	是	否	否	否

① 首先，随机选择一个特征项，比如“是否有房”，作为决策树的根节点，

建立决策树如图 4.1 所示（后续的“分枝选择问题”中会讲述如何选取属性，以获取更好的分类效果）。图中数字标明了“有房”和“无房”的客户拖欠贷款的人数。

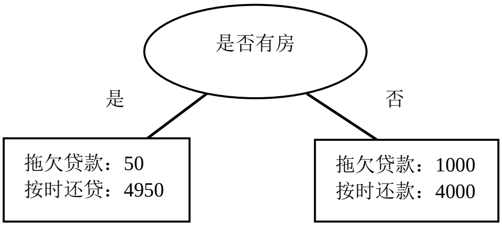


图 4.1“是否有房”决策树分枝

② 如果有房，则按时还款率高达 99%。说明如果客户有房子，银行可以放心贷款，不需要再考虑其他因素。因此，决策树模型在该分枝可以终止拆分；无房的客户按时还款率只有 50%，银行需要综合其他因素做出决策，因此，需要从该分枝继续拆分。此处，选取“是否已婚”属性作为分枝依据，拆分结果如图 4.2 所示。

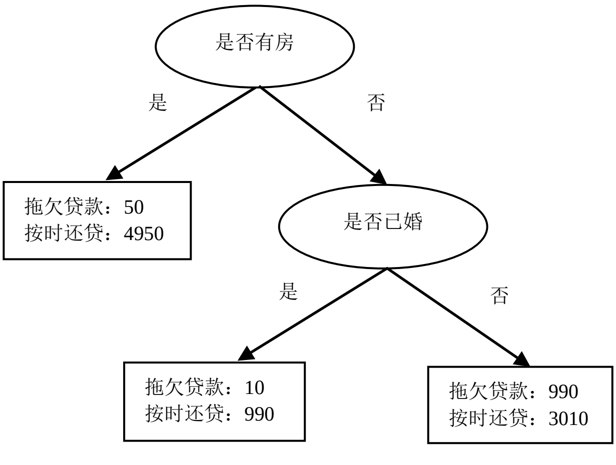


图 4.2 “是否已婚”决策树分枝

③ 对于无房但已婚的客户，按时还贷的概率有 99%，因此终止该分枝的继续分裂。依据同样的步骤，选取其他特征逐步划分，直到节点的“纯度”达到要求，或者所有的属性都已划分完毕，则终止决策树的创建。

（2）分枝选择问题

在上述决策树模型的构建过程中，根节点以及中间过程中的分裂节点是随机选择的。显而易见，数据集中必然有些特征值对分类所起的效用更高，有些则偏低。优先选取这些对分类效果影响更大的特征值分裂，才能构建更加合理的决策树模型。那么如何考量各个特征值对分类效用的高低?

依据某特征值分裂后，利用节点的信息“纯度”来说明该特征值对分类影响的大小。所谓信息“纯度”是指节点中数据类别的一致性程度。纯度越高，说明其包含的信息越少，数据类别的一致程度越高。选取分裂后获得的节点“纯度”最高的特征值进行分裂。度量“纯度”高低的常用指标有 Entropy（信息熵）、Gini（基尼系数）、Error（错误分类率）、InfoGain（信息增益）、InfoGainRatio（信息增益率）[2]。不同的决策树可能选择不同的指标选取分裂特征。我们以银行借贷举例，分别解释各考量指标的计算方式。

图 4.3 中对 10,000 个银行借贷用户，分别依据“是否有房”和“是否已婚”两个特征值进行树的分裂。两个特征值谁包含的信息量大，就优先选择这个特征值进行划分，我们将在下述的介绍中逐次说明。

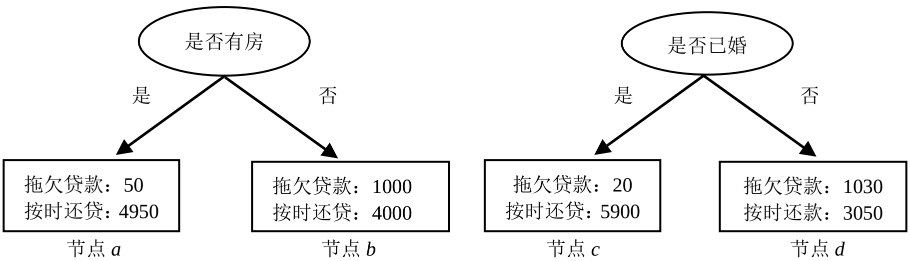


图 4.3　不同特征分裂结果图

① 信息熵

$$\text{Entropy}(t) = -\sum_{i=0}^{c-1} p(i|t)\log_2 p(i|t) \qquad (4.1)$$

其中 c 是类的个数，t 代表某个分裂节点。根据该公式，计算图 4.3 各节点的信息熵如表 4-2 所示。

表 4-2　信息熵的计算结果

节　　点	信息熵
a	$-0.01\times\log_2 0.01\ -0.99\times\log_2 0.99=\ 0.0808$
b	$-0.2\times\log_2 0.2\ -0.8\times\log_2 0.8=\ 0.7219$
平均信息熵	$0.5\times 0.0808\ +\ 0.5\times 0.7219\ =\ 0.4014$
c	$-0.0033\times\log_2 0.0033\ -0.9967\times\log_2 0.9967=\ 0.0319$
d	$-0.2525\times\log_2 0.2525\ -0.7475\times\log_2 0.7475=0.8152$
平均信息熵	$0.592\times 0.0319\ +\ 0.408\times 0.8152\ =\ 0.3503$

“是否有房”分裂后的平均信息熵是 0.4014；“是否已婚”分裂后的平均信息熵是 0.3503。选择信息熵较低的节点进行分裂，可以获得更高的节点“纯度”，划分效果更好。因此，“是否已婚”特征值的优先级比“是否有房”高，应该优先选择。

② 基尼系数

$$\mathrm{Gini}(t)=1\ -\sum_{i=0}^{c-1}[p(i|t)]^2 \tag{4.2}$$

其中 c 是类的个数，t 代表某个分裂节点。根据该公式，计算图 4.3 各节点的基尼系数，如表 4-3 所示。

表 4-3　基尼系数的计算结果

节　　点	基尼系数
a	$1\ -0.01^2\ -0.99^2=\ 0.0198$
b	$1-0.2^2\ -\ 0.8^2=0.32$
平均基尼系数	$0.5\times 0.0198\ +\ 0.5\times 0.32\ =\ 0.1699$
c	$1-0.0033^2-0.9967^2\ =\ 0.0066$
d	$1-0.2525^2\ -0.7475^2=0.3787$
平均基尼系数	$0.592\times 0.0066\ +\ 0.408\times 0.3787\ =\ 0.1584$

与信息熵计算的结果一致。“是否已婚”的平均基尼系数小于“是否有房”的平均基尼系数，说明前者含有的信息量低。所以应该优先选择“是否已婚”这一特征值来分裂。

③ 错误分类率

$$\text{Classification Error}(t) = 1 - \max([p(i|t)])\{i = 1..c\} \quad (4.3)$$

其中 c 是类的个数，t 代表某个分裂节点。根据该公式，计算图 4.3 各节点的错误分类率如表 4-4 所示。

表 4-4 错误分类率的计算结果

节 点	基尼系数
a	$1 - 0.99 = 0.01$
b	$1 - 0.8 = 0.2$
平均错误分类率	$0.5 \times 0.01 + 0.5 \times 0.2 = 0.105$
c	$1 - 0.9967 = 0.0033$
d	$1 - 0.7475 = 0.2525$
平均错误分类率	$0.592 \times 0.0033 + 0.408 \times 0.2525 = 0.1049$

从错误分类率的角度来看，与其他度量指标的结论一致。应该优先选择“是否已婚”分裂。

④ 信息增益

$$\text{InfoGain} = \text{Entropy(parent)} - \text{avgEntropy(child)} \quad (4.4)$$

信息增益等于父节点的熵与子节点的平均信息熵的插值。应当选取使得信息增益最大的特征值分裂。但是，无论是使用信息增益还是信息熵来度量节点的纯度，在属性分类较多的情况下都可能存在问题。比如，在用户按时还贷的决策树算法构建中，用户 ID 是系统自动生成的编号，对该用户是否能按时还贷没有任何关联。假设按照借贷用户的 ID 属性进行决策分类，那么每个分类中只有一个用户，其信息“纯度”都是 100%。但是，用这种方式构建决策无任何意义。类似这样的属性应该从特征项中剔除，或者不应赋予高优先级。按照用户 ID 属性分裂的结果如图 4.4 所示。

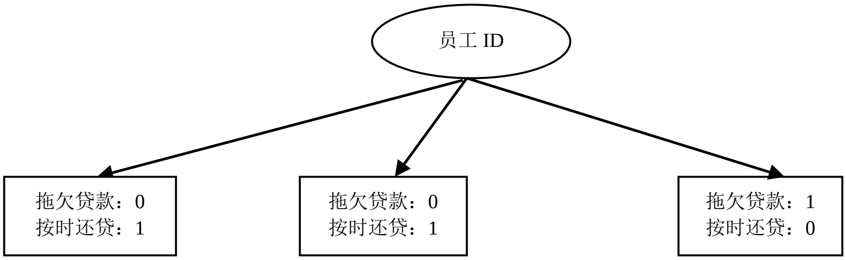


图 4.4　依据员工 ID 的分类结果

由图 4.4 可知，以 ID 为特征值分裂，其节点的熵最小，信息增益最大，应该作为决策树的根节点。但是，这种分裂显而易见并不合理。因此，我们引入下述的“信息增益率”解决此类问题。

⑤ 信息增益率

$$\text{InfoSplit}（D|A） = -\sum_i^n \frac{|D_i|}{|D|} \times \log_2(\frac{|D_i|}{|D|}) \tag{4.5}$$

$$\text{InfoGainRatio}（t） = \frac{\text{InfoGain}(A)}{\text{InfoSplit}(D|A)} \tag{4.6}$$

其中 *A* 代表属性，*D* 代表节点数据的个数，*n* 代表分裂节点。InfoGainRatio（信息增益率）=信息增益 / 信息分裂值。

以图 4.3 所示的数据为例，计算“是否有房”与“是否已婚”的信息增益率如表 4-5 所示。计算信息增益率，首先需要计算父节点的熵。假设首次对决策树进行分裂决策，该父节点的熵，就是指整个训练数据集的熵。

$$\text{父节点熵} = -0.105 \times \log_2 0.105 - 0.895 \times \log_2 0.895 = 0.4846$$

表 4-5　信息增益率

度量指标	是否有房
InfoGain	$0.4846 - 0.4014 = 0.0832$
InfoSplit	$-0.5 \times \log_2 0.5 - 0.5 \times \log_2 0.5 = 1$
InfoGainRatio	$\frac{0.0832}{1} = 0.0832$

续表

度量指标	是否已婚
InfoGain	$0.4846 - 0.3503 = 0.1314$
InfoSplit	$-0.592 \times \log_2 0.592 - 0.408 \times \log_2 0.408 = 0.9754$
InfoGainRatio	$\frac{0.1314}{10.9754} = 0.1347$

比较两个属性的信息增益率，显而易见，“是否已婚”的信息增益率较高，其分裂优先级高于“是否有房”，在构建决策树时应优先选择。

熵、基尼系数、错误分类率和信息增益率等度量指标的区别在于看待问题的角度不同，分别从一定的概率上说明哪些选择更优。如果最终的分类效果均不太理想，还应该综合其他各种因素，做进一步的考虑。

（3）剪枝终止问题

决策树建模算法是逐步选择合理的特征值进行分裂，直到所有的特征值都用于分裂，或者某节点的信息“纯度”达到预设的指标，则终止树的构建。工程师希望模型具有如图 4.5 所示的分类决策边界，在训练集和测试集上都可以达到比较理想的分类效果。

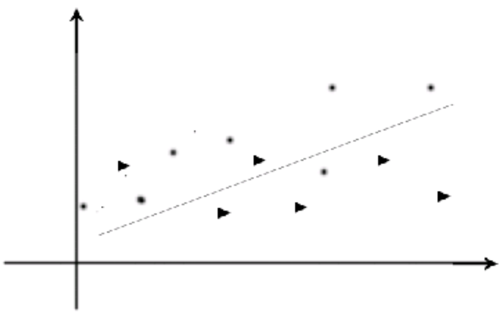


图 4.5　正常拟合示意图

但是，特征值的选取，或者数据模型参数调整有误，会给模型造成“欠拟合”的问题。所谓“欠拟合”是指决策树的拟合模型欠佳，在训练集和测试集都表现得不理想。其分类的效果如图 4.6 所示。

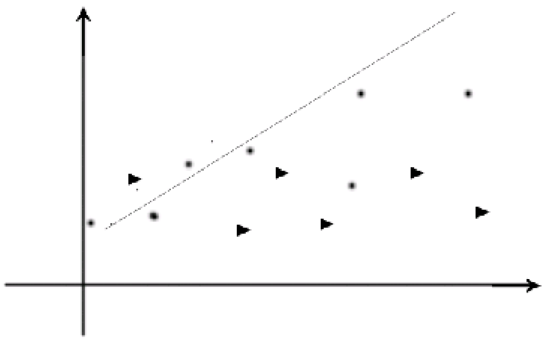


图 4.6　欠拟合示意图

为了使模型能够获取更好的分类效果，工程师往往选取更多的特征值，或者让决策树一直分裂下去，直到用尽所有特征值。然而，通常情况下由于划分太细，又极有可能导致最终所构建的模型出现“过拟合”现象。所谓“过拟合”是指决策树模型在训练集上有完美的分类效果，但是在其他测试集合上却表现糟糕。训练集上的分类效果如图 4.7 所示。

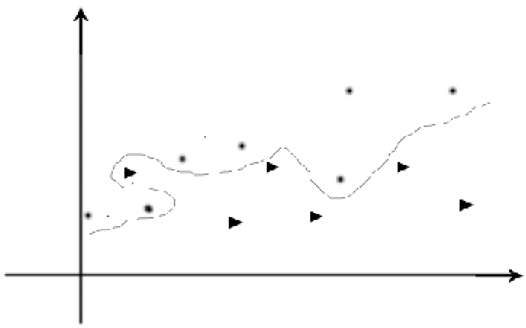


图 4.7　过拟合现象

剪枝策略能较好地避免“过拟合”问题。剪枝分为“先剪枝”与“后剪枝”两种。前者在树的构建过程中，设置剪枝阈值，当信息熵或者信息增益等不纯性度量的指标低于设定阈值时就停止该分枝的增长。“先剪枝”的优势在于其可以及时终止构建复杂的子树；同时也一定程度提高模型的泛化能力；然而，其缺点在于很难恰如其分地设定阈值。“后减枝”策略是在完全构建完成的决策树的基础上，自底向上剪去不纯性度量值低于设定阈值的子树，从而提高模型的泛化能力。和

“先剪枝”相比，由于它是在完整构建决策树后进行剪枝的操作，所以能在整体上做出更好的剪枝决策；然而，它的缺点是需要构建完整的决策树，浪费了一定的计算能力。因此，选取合适的剪枝策略，并不断调整剪枝的阈值，在“过拟合”和“欠拟合”中寻求更优的参数，才能让模型具有更强的泛化能力。

（4）连续数据问题

上述构建决策树的训练集的特征值都是二元离散值，使用“是”或“否”即可实现树的分裂生长。如果特征值中有连续的数据，那么决策树又该如何分裂构建呢?

我们举例说明。以“工资”为例，假设把“工资”这一属性，按 $1,2,3,\cdots,n$ 以及大于 n 的共 $n+1$ 个分枝进行分裂，这种分裂方式计算量巨大，同时对于分类的效果收益甚微。在决策树的构建过程中，对连续特征值的分裂通常可以采用以下方法。

① 将训练数据集中某连续特征值从小到大进行排序。

② 取特征值发生改变的点作为分割点，获取分裂的数值 S_1、S_2、$S_3,\cdots,S_m$。

③ 将小于 S_1 的数值划分为同一分枝，大于 S_1 小于等于 S_2 的数值划分为同一分枝，依此类推。

表 4-6　按工资排序的训练集

用户 ID	工资（元）	是否有房	是否有车	是否已婚	本科及以上	拖　欠
300	2500	是	否	是	是	否
49	2500	否	是	否	否	是
72	2500	是	是	否	否	否
1800	3000	是	否	是	否	否
11	5000	否	否	是	是	是
996	5000	是	是	是	是	否
87	7100	否	是	否	是	否
…	…	…	…	…	…	…
6		是	是	否	否	否

由表 4-6 可以看出，“工资”的划分点分别有 2500、3000、5000 等。对于其

他的连续属性，可类比对“工资”属性进行分类的方式，逐步找出分裂的数值点。

（5）ID3、C4.5、CART 的比较

依据不同的分裂策略、剪枝策略，是否支持连续属性等特点，又能构建出不同的决策树。其中，ID3、C4.5 以及 CART 是三种经典的决策树建模方案。

CART 由 L.Breiman、J.Friedman、R.Olshen 和 C.Stone 在 1984 年提出，既可以作为分类树也可以作为回归树（回归树本章没有讲解，读者可先忽略）。当它为分类树时，采用基尼系数作为不纯度度量，使用“后剪枝”策略；同时，既可以处理连续数据，也可以处理离散数据。CART 只对属性进行二元划分。如果某离散特征的取值超过两类，则需要选取使基尼系数最大的特征，以“是”或者“非”的方式分裂；如果是连续属性，则以均值为衡量，把“大于等于”和“小于”该均值的数据分裂成两类。因此，CART 决策树不适用于离散特征有多个取值的场景。

ID3 由 RossQuinlan 在 1986 年提出。它采用信息增益作为不纯度度量，选取当前信息增益最大的属性划分。树的构建不考虑剪枝，同时不支持连续数据和缺失值，所以构建过程简单且训练速度快。

C4.5 由 RossQuinlan 在 1993 提出。它在 ID3 的基础上做了改进，实现了对连续数据的支持，采用“先剪枝”的剪枝策略，同时支持对缺失值的处理。它是当前最常用的决策树算法。

下面我们从不纯度度量的选择、剪枝策略的选择、是否能处理连续数据比较以上决策树，如表 4-7 所示。

表 4-7　ID3、C4.5、CART 算法比较

算　　法	不纯度度量	连续数据	剪枝策略	分枝划分	缺失值
ID3	信息增益	不支持	无剪枝	多分	不支持
C4.5	信息增益率	支持	先剪枝	多分	支持
CART	基尼系数	支持	后剪枝	二分	支持

2．朴素贝叶斯分类

（1）模型的构建

朴素贝叶斯分类是从概率的角度来看待分类的问题。即当某些特征出现时，数据为哪一类的概率最大，则认为其属于该类。用数学公式表示如下。

$$P(Y=y_1|X=(x_1,x_2,x_3,\cdots,x_n)) \tag{4.7}$$

也就是当特征向量 X 取值为 x_1, x_2, ⋯，x_n 时，Y 属于类 y_1 的概率是多少。取概率最大的类作为该数据的类别。假设我们有足够的训练数据集，则可以计算出在 X 所有可能取值的情况下，Y 属于各个类的概率，并把 X 的所有可能取值及其对应的类别 Y 的概率做记录留存。后续对于输入的数据，可以通过如下方式预测其类别：首先提取数据的特征向量；其次通过哈希等方式，从记录中找出所对应各个类别的概率，取概率最大的类作为该数据样本的类别。

然而，上述思路存在一些问题。比如，有 20 个特征值，每个特征值用 0 或 1 表示发生或者不发生，那么就有 2^{20} 种组合，同时，特征向量的组合随着特征值的增加呈指数型增长，因此，很难有足够的数据和计算能力以穷举的方式求解每种特征向量属于各个类别的概率。此时，依据贝叶斯条件概率，有如下公式。

$$P(Y|X)=\frac{P(X|Y)P(Y)}{P(X)} \tag{4.8}$$

对于某特征向量，分别求出其属于各个类别的概率，即 $P(y_i|X)$的值。不难发现，当我们对比该样本属于 A 类还是 B 类时，$P(X)$值都是一样的，因此，整个朴素贝叶斯分类的问题就转化为求解 $P(X|Y)$、$P(Y)$的概率值。

$P(X|Y)$的求解思路就是在 $Y=y_i$ 的情况下，先判断 X 属于哪种概率分布，进而针对特定的概率分布，求解概率公式，最终依据概率公式获得概率值。下面我们要讲的多元伯努利模型、多项式模型、高斯模型是根据 X 不同的概率分布所形成的三种朴素贝叶斯分类模型。

（2）多元伯努利模型

我们以文本分类为例说明朴素贝叶斯分类的多元伯努利模型。假设有训练数据集如表 4-8 所示，其中 0、1 代表文章中是否有该词出现，0 代表没有，1 代表有。

表 4-8　多元伯努利模型数据训练集

ID	大侠	刀剑	秘籍	比武	相对论	宇宙	黑洞	光年	类别
1	1	1	1	0	0	0	0	0	武侠
2	0	0	1	0	1	1	1	1	科普
3	0	0	0	0	1	1	1	0	科普
4	0	1	1	1	0	0	0	1	武侠
5	0	0	1	0	0	1	1	1	科普
…									
N	0	0	0	0	1	1	1	1	科普

如果某文章中出现了“大侠”“刀剑”“秘籍”“比武”，没有出现“相对论”“宇宙”“黑洞”“光年”时，该文章属于“科普”还是“武侠”？形式化表示该问题就等于是求解 $P(X=(1, 1, 1, 1, 0, 0, 0, 0) \mid Y=$“武侠”$\times P(Y=$“武侠”$)$和 $P(X=(1, 1, 1, 1, 0, 0, 0, 0) \mid Y=$“科普”$) \times P(Y=$“科普”$)$的概率，取值较大的作为其类别。由于该种场景中，特征向量 X 属于多元伯努利概率分布，因此$P(X) = \prod_{i=1}^{n} x_i$，所以上述朴素贝叶斯的概率公式转换为如下形式。

$$P(X|Y) = \prod_{i=1}^{n} P(x_i|y_k) \tag{4.9}$$

利用训练集中的数据，通过简单统计最终计算出 $P(X=(1, 1, 1, 1, 0, 0, 0, 0) \mid Y=$“武侠”$)$和 $P(X=(1, 1, 1, 1, 0, 0, 0, 0) \mid Y=$“科普”$)$的概率。

（3）多项式模型

在多元伯努利模型中，某类文章的生成过程，可以看成是在一堆词汇中不断选择或舍弃。然而，这种方式丢失了每个词汇选取次数的重要信息，因此我们需要换个角度来看待此问题。此时，我们可以想象有很多供选取的独立词汇，各个词汇的选取次数最终决定文章类别的归属。仍以文章分类为例，生成的训练集如

表 4-9 所示。

表 4-9　多项式模型数据训练集

ID	大侠	刀剑	秘籍	比武	相对论	宇宙	黑洞	光年	类别
1	12	34	73	123	0	3	0	6	武侠
2	0	0	12	3	53	21	64	91	科普
3	0	1	3	0	14	71	14	3	科普
4	0	11	18	41	0	0	0	1	武侠
5	6	2	1	0	0	18	15	13	科普
…									
N	0	1	0	0	11	31	5	1	科普

求解 $P(X=(12，12，32，12，1，3，0，0)|Y=$“武侠”)和 $P(X=(12，12，32，12，1，3，0，0)|Y=$“科普”)的概率，即可判别特征向量 $\boldsymbol{X}$=(12，12，32，12，1，3，0，0)的文章的类别。形式化描述该问题则是文章的训练集由{大侠、刀剑、秘籍、比武、相对论、宇宙、黑洞、光年}这八个典型的词汇挑选组合而成。Y 属于武侠类文章时，每个特征值被选取的概率分别为{p_{m1}，p_{m2}，p_{m3}，p_{m4}，p_{m5}，p_{m6}，p_{m7}，p_{m8}}，其中 P_{mi}=某词出现的个次数 / 训练集中总的词汇个数；同理，Y 属于科普类文章时，每个特征值选取的概率分别为{p_{n1}, p_{n2}, p_{n3}, p_{n4}, p_{n5}, p_{n6}, p_{n7}, p_{n8}}。X=(12，12，32，12，1，3，0，0)说明该篇文章总共进行了(12 + 12 + 32 + 12 + 1 + 3 +0 +0)=71 次选词。此时，问题转换为当 Y 属于武侠类文章时，在 71 次选词过程中，选取 {大侠、刀剑、秘籍、比武、相对论、宇宙、黑洞、光年}的次数依次为{12, 12, 32, 12, 1, 3,0,0}的概率；以及当 Y 属于科普类文章时所对应的概率。该场景是典型的多项式概率事件，可以依据多项式概率分布求解概率公式，进而对类别做出判断[13]。

（4）高斯模型

对于特征向量中有连续属性的场景，引入朴素贝叶斯的高斯模型。仍以决策树中银行信贷的示例作为说明。假设有训练数据集如表 4-10 所示。

表 4-10　高斯模型数据训练集

用户 ID	工资（元）	是否有房	是否有车	是否已婚	本科及以上	拖　欠
1	1200	是	否	是	是	否

续表

用户 ID	工资（元）	是否有房	是否有车	是否已婚	本科及以上	拖　欠
2	1400	否	是	否	否	是
3	2300	是	是	否	否	否
4	4500	是	否	是	否	否
5	11200	否	否	是	是	是
6	23000	是	是	是	是	否
7	7100	否	是	否	是	否
…	…	…	…	…	…	…
N		是	是	否	否	否

由于“工资”这种连续属性的存在，使得无法使用上述的多元伯努利模型实现概率的计算。通常情况下，工资的概率分布属于高斯模型，那么工资数额等于某值的概率，可以用以下公式求出。

$$\int_{x}^{x+\varepsilon}\frac{1}{\sqrt{2\pi\delta}}\mathrm{e}^{(-\frac{(x-\mu)^2}{2\delta^2})} \tag{4.10}$$

其中ε是一个大于 0 的极小值。该问题转换为多元伯努利模型求解，本质上是求解公式 4.8。可以使用公式 4.10 来计算“工资”这一特征值的概率，其中ε会相互抵消。高斯模型其实就是对多元伯努利模型的改进，以解决计算连续属性概率的问题。

（5）限制条件及拉普拉斯平滑

朴素贝叶斯模型的推导有两个限制条件：① 特征向量各特征值之间是相互独立的；② 各个特征值的权重相当。虽然有时这两个条件并不是严格成立，但是朴素贝叶斯模型依然有很好的分类效果。

与此同时，在计算概率时，可能遇到某一个特征值的概率为 0。比如在多项式模型的示例中，假设训练集中所有的科普类文章都没有出现“黑洞”这一特征值（可能是由于数据集不全导致的），而某篇待预测的文章有“黑洞”一词，则在计算推导时，$P(x_i$=“黑洞”$|Y$=“科普”$)=0$，最终概率结果为 0。为了避免这种情况，往往引入拉普拉斯平滑，又称为“加一平滑”，使得朴素贝叶斯的最终计

算公式如下：

$$P(Y|X) = \frac{P(X|Y)P(Y)+1}{P(X)+|V|} \tag{4.11}$$

其中|*V*|是特征向量的可能取值的种类。在多元伯努利模型中|*V*|的值就是 2，因为随机变量只有“选取”或者“舍弃”两种取值；同理，在多项式模型的例子中|*V*|的值是 8，即有 8 种不同的词可供选择。

3．KNN 规则

（1）模型的构建

KNN（K-Nearest Neighbor）是一种用于数据分类、有监督的建模方法。与决策树、朴素贝叶斯相比，是较为常用且十分简单的模型。所谓 KNN，就是根据事先拟定的距离度量公式，计算某未知类别点与 *K* 个相邻数据点的距离；如果某种类别拥有的数据点个数最多，那么未知点就属于该类别。算法流程如下。

① 假设数据集中有一定数据量的点，已经标注类别。
② 对待判定的点，根据拟定的距离，计算与其最近的 *K* 个点。
③ 根据 *K* 个“邻居”中类别，来判断该点的类别。

下面用图形的方式做直观的阐述，如图 4.8 所示。

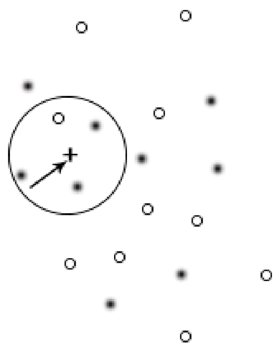


图 4.8　KNN 算法示意图

如图 4.8 所示，实心圆和空心圆分别代表数据集的 A、B 两类数据。箭头所指的“*x*”代表某未知类别的样本，既可能是 A，也可能是 B。根据 KNN 建模，假设 *K* 的取值是 4，由于与“*x*”邻近的 4 个点中，有三份样本的类型是 A，因此，“*x*”应该是 A 类别。

决策树、朴素贝叶斯等模型，在训练集中求解得到模型之后，只需把待分类的数据代入，即可求解类别；而 KNN 模型每次都需要计算数据集才能得到样本的类别，这在某些需要即时分类的场景中是不适用的。同时，KNN 建模中距离的选择也至关重要，这方面的内容，将在 4.2 节中详细阐述。

4.1.2 聚类

聚类，可以在不标注训练数据集类别的前提下，让数据依据类别聚合。与分类算法相比，这种无需事先标注训练集类别的学习方式，被称为非监督学习。依据聚类方法不同，又可以划分为四种类型：划分聚类、层次聚类、密度聚类以及模型聚类。本节将依次讲解各模型的建模流程以及应用场景。

1．划分聚类

（1）模型的构建

k-means 属于典型的划分聚类算法。依据业务场景，当已知数据集具有 *K* 个类别时，*k*-means 算法可以有效地划分各类数据（*k*-means 必须已知数据集的类别）。首先，已知有 *K* 个类别（一般来自实践经验和业务场景）；其次，在训练集上运行 *k*-means 算法，且依据聚合效果，不断尝试调优参数，这些参数泛指特征项的选择如 *K* 的个数等；最后在获得较为满意的结果之后，把调参后的模型用于测试集测试，并根据相关指标评估。*k*-means 算法的建模流程如下。

① 随机选择 *K* 个点作为初始的质心。

② 计算各点到质心的欧几里得距离（距离平方和），并指派到最近质心，形成 *K* 个簇。

③ 重新计算每个簇的质心。

④ 重复步骤 2、步骤 3 直到质心不发生变化。

如图 4.9 所示，假设数据集中混有三类数据，*k*-means 聚类的训练过程如下。

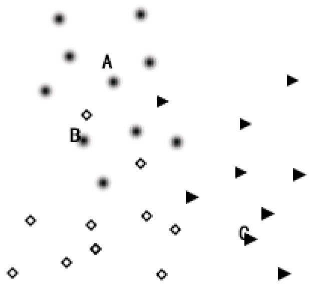


图 4.9(a)　*k*-means 聚类过程示意图

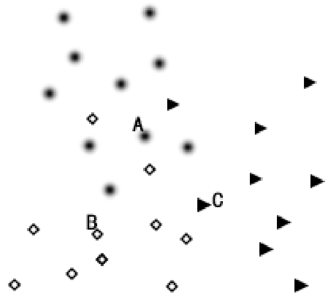


图 4.9(b)　*k*-means 聚类过程示意图

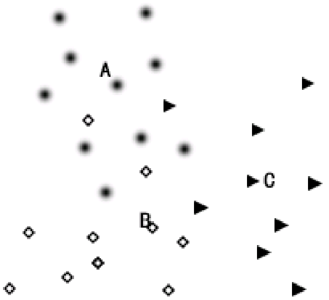


图 4.9(c)　*k*-means 聚类过程示意图

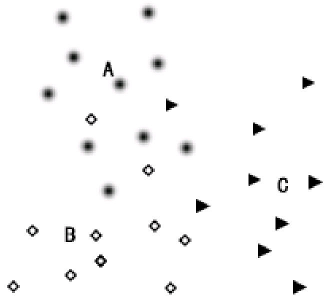


图 4.9(d)　*k*-means 聚类过程示意图

首先随机指定三个质心，经过如图所示 a→b→c→d 的不断迭代，质心不断迁移，直到最终达到稳定状态，迭代终止。最后，判断各点与质心的距离，把具有相同质心的样本归为同一簇，即可获得最终的聚类结果。

k-means 算法的第三步中，选取数据簇中各点的维度“均值”作为质点，这也是“*k*-means”的含义。比如(5,3,7),　(1,4,9),　(12,1,3)的质点就是((5+1+12) / 3, (3 + 4 + 1) / 3,　(7 + 9 + 3) / 3)所表示的点。因此，质心不一定是数据集中的点。

显而易见，虽然通过对训练集的训练可以获取聚类模型参数；但是对于待聚类的数据集合，仍然需要执行聚类过程，才能得到最终的聚类结果。

根据“初始点选取的方式”“离群值问题的处理”“离散数据的处理”“非凸数据的处理”的不同，*k*-means 又有 *k*-means++、*k*-medoid、*k*-medians、*k*-modes 等变种。下面会逐步由问题引出相关的算法。

（2）初始点问题

上述 *k*-means 算法对初始值的设置十分敏感。其初始质心的选择直接影响了迭代计算的路径，从而最终影响到聚类效果的好坏。选择距离尽可能远、相似度尽可能低的点作为初始点，对模型的训练至关重要。*k*-means++算法是在 *k*-means 的基础上，调整初始点选取的方式，其建模流程如下。

① 假设需要指定 K 个质点,首先从输入数据集合中随机选一个点做初始点。
② 计算其余各点到该初始质点的距离，设为 $D(x)$。
③ 以“$D(x)$越大的点，选取的概率较高”的策略选取第二个质点。
④ 重复步骤 2、步骤 3 直到选取 K 个质点。
⑤ 利用标准 *k*-means 算法聚合。

步骤 3 中，采用如下方法使 $D(x)$越大的点被选为质点的概率越高。

① 计算其余的数据点，与已选质心的距离（第一个质心是随机选出的）。
② 记录离最近质心的距离，记为 $D(x_i)$，所有的 $D(x_i)$相加，$\mathrm{Sum}(D(x))$。
③ 选取能落在 $\mathrm{Sum}(D(x))$的随机值，记录为 Random。
④ 依次计算 $\mathrm{Random}-D(X)$，直到差值< 0，则该点为新的质心。
⑤ 重复步骤 2、步骤 3 和步骤 4，直到选出第 K 个质心。

相比于 *k*-means 算法，*k*-means++能更好地设定初始质心，从而有效缩短迭代周期，获取更好的聚类效果。

（3）离群值问题

由于 *k*-means 采用“欧几里得距离”度量数据点之间的距离，这使得在计算质心的过程中，如果存在离群值和异常值，聚类效果将不尽人意。

如图 4.10 所示，箭头标注的“黑三角”即为离群值。如果按照图 4.9 的迭代过程来构建模型，当数据集中存在这种离群值时，会导致聚类质心点的偏离，影响整个聚类模型的效果。针对此类问题，在 *k*-means 的基础上提出了 *k*-medoid、*k*-medians 算法，简述如下。

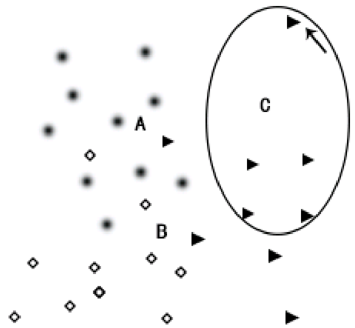


图 4.10　具有离群值的数据集

如前面讲解 *k*-means 中所述，它选取均值作为质心点，该均值并不一定是数据集中具体的某个点；*k*-medoid 在更新质心的过程中，选取一个真实存在的数据点，该点到该簇中的其余各点的距离平方和最小，其他步骤则与 *k*-means 相同。

k-medians 选取各维度的中位数作为该簇的质心。同时，各点与质心的距离采用“曼哈顿距离”，而非欧几里得距离。比如有 *a*(10, 1, 5), *b*(3, 31, 8), *c*(12, 3, 4)，则 *k*-medians 算法中其质心为(10, 3, 5); 同时 *a* 到质心的距离为(10 – 10, 3 – 1, 5 – 5)。“曼哈顿距离”的概念也会在后面的“杂项”中详细讲解。

（4）非数值问题

无论是 *k*-means 算法、*k*-medoid 算法，还是 *k*-medians 算法，它们的数据集特征值必须是数值类型。为了解决这个问题，提出了 *k*-modes 算法，它是在 *k*-means

的基础上对非数值类型聚类建模的方式。它的步骤如下。

① 随机选择 *K* 个点作为初始的质心。

② 将每个点指派到最近的质心。该点与质心对应的非数值属性如果相同，则规定它与质心的距离为 $D(x_i)+1$。依次把所有点划分到 *K* 个簇中。

③ 在每个簇中，依次选取各维度出现频次最高的属性值，组合该簇的质心。

④ 重复步骤 2、步骤 3 直到质心不发生变化。

我们举例说明。假设有由 *a*（已婚，本科，本地人）、*b*（已婚，本科，外地人）、*c*（未婚、硕士、外地人）三个数据点组成的簇，其中第一维中“已婚”出现最多，第二维中“本科”出现最多，第三维中“外地人”出现最多，因此，选取（已婚、本科、外地人）组成的特征向量作为该簇的质心。

（5）簇的形状

k-means 和与之相关的聚类建模方法，对数据集簇的形状都有一定的要求，我们用图 4.11 说明。

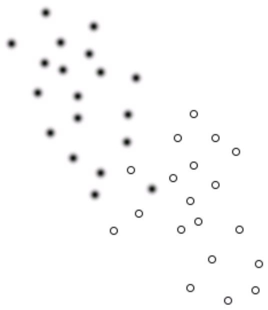


图 4.11(a)　不同形状的数据集族

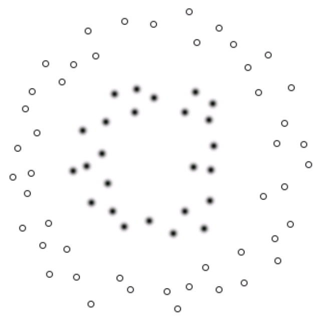


图 4.11(b)　不同形状的数据集族

图 4.11(a)类型的簇可以更好地使用 *k*-means 及其改良算法；如果直接对形如图 4.11(b)的数据簇进行 *k*-means 聚类建模，效果则不甚理想。对于图(4.11)b 类型的数据簇，可以使用 kernel *k*-means，将数据集映射到高纬空间，之后进一步实现 *k*-means 算法。此处不再赘述，感兴趣的读者可以参阅相关文献。除此之外，接下来我们要讲述的 Dbscan 密度算法，则能对包括图 4.11(b)在内的任意形状的簇进

行聚类。

上述的 *k*-means 及其改良算法是典型的划分聚类，属于无监督学习，即数据集不需要事先做类别标注。虽然原始的 *k*-means 算法存在诸多问题，但其模型的训练速度却出类拔萃。因此，要依据实践的场景，从建模的时间、是否存在异常值、特征值是否非数值型等多个维度，酌情地选择合适的算法。

2．层次聚类

（1）模型的构建

k-means 对不重叠的、具有“对等”关系的类进行划分，而层次聚类则用于对“嵌套”关系的类做聚合。数据集中需要聚合的类别，属于彼此对等的，还是彼此嵌套的，决定了聚类是层次聚类还是划分聚类。例如，对一批新闻数据进行聚类，将其聚合为{体育新闻、娱乐新闻、经济新闻、其他}四类，这属于划分聚类；如果按照{中国新闻、体育新闻、足球新闻、篮球新闻、娱乐新闻、影视娱乐、音乐娱乐}进行聚类，则它属于层次聚类。划分聚类算法实现对不重叠的子集的划分，而层次聚类则着眼于处理嵌套的类别，形成树状的嵌套分类集合。层次聚类获取的聚类结果如图 4.12 所示。

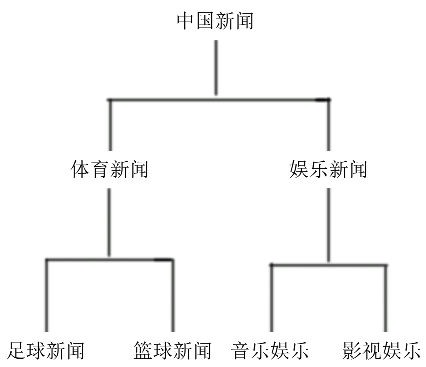


图 4.12　层次聚类的聚类结果

层次聚类的建模过程，可以分为自上而下和自下而上两种方式。其中自上而下的方式使用的频度并不高，因此仅介绍自下而上的建模过程，流程如下。

① 首先每个样本都视为一类，N 个样本就代表 N 类。

② 计算每个类与类之间的距离，选择距离最近的归为一类。

③ 重新计算各个类与其他类之间的最近距离。

④ 重复步骤 2、步骤 3，直到所有的样本都归为一类。

该建模过程由单个样本自下而上开始，层层聚合，最终形成如图 4.12 的层次结构。当需要获取某一子类时，只需要探索合适的层高，并获取该节点及其子节点即可。

（2）类间距选择

模型构建的第 2 步中类与类之间的距离计算有四种模式，分别如下。

① Single-Linkage：取两类中距离最近的样本作为类之间的距离。该方法对存在异常点的情况效果较差。比如包含有多个样本的两个类中，只有两个点距离比较近，其余各点都距离很远，那么该方式效果较差。

② Complete-Linkage：与 Single-Linkage 正好相反，选取类与类之间样本距离最大值作为距离。同样也存在和 Single-Linkage 相同的问题。

③ Average-Linkage：采用各类中样本之间距离的均值作为类之间距离的度量。其整体效果优于上述两种方式，如果存在影响较大的异常点，依然存在类似问题。

④ Median-Linkage：采用类与类之间样本距离的中位数作为类的距离，可以抗拒异常点的干扰。

3．密度聚类

（1）模型的构建

Dbscan 是基于密度的聚类算法，利用数据点的稠密程度来划分类别，数据点越聚集越应该聚集成一类。介绍算法之前，我们先了解几个比较重要的概念，如图 4.13 所示。

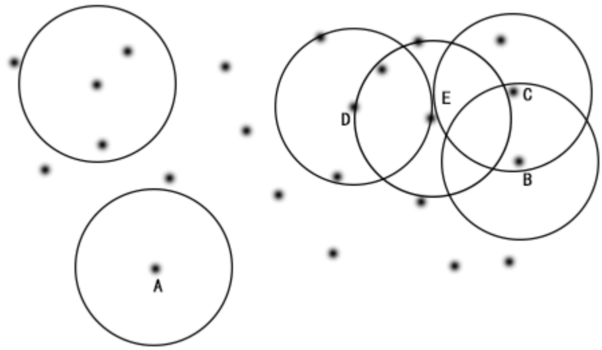


图 4.13　Dbscan 算法示意图

C 点称为核心点：即以某一点为中心，以给定 Eps 的半径画圆，如果该圆内有超过 MinPts 个邻近点包含在其中，那么该中心点即为核心点。其中 Eps 和 MinPts 都是用户指定的值，分别代表半径，以及在该半径内数据点个数的阈值。该图中 Eps = 3, MinPts = 3。

B 点称为边界点：某一点不满足核心点的定义，但是落在核心点的圆形区域内，那么该点就是边界点。边界点可能落在多个核心点的邻域内。

A 点称为噪声点：既非核心点也非边界点的点被称为噪声点。

Dbscan 的算法流程如下。

① 依据上述定义，把数据集中的每个点分别标记为核心点、边界点、噪声点。
② 删除所有的噪声点。
③ 连通在 Eps 半径距离之内，相互交叉的所有核心点，其形成的簇归为一类。
④ 指派边界点到其被包含的簇中。

（2）Eps 和 MinPts 的选择

Eps 和 MinPts 的选择没有确定的方法，通常要凭借对数据集的认知、过往的经验以及最终聚类的结果来不断调整参数；同时，引入 *k*-距离的方法来指导参数的选取。

在选取 *k*-距离时，我们首先假定 MinPts=*k*，最初的 Dbscan 算法中 MinPts 取值是 4（当然也可以选取其他数值）；其次，计算某点到其余各点的距离，将它们按递增顺序排序，并取排在第 *k* 个位置上的值，称为 *k*-距离；之后，将各个点的 *k*-距离按递增顺序排序，并对排序后的各个数据点以自然数 1 开始标号；最后，以数据点序号为 *X* 轴，*k*-距离为 *Y* 轴绘制图形。找出所绘制的曲线中发生急剧变化的点，并将它所对应的 *k*-距离作为 Eps。

（3）适用场景

与 *k*-means 聚类相比，Dbscan 聚类不需要事先输入聚类的个数，适用于任何形状的簇，对噪声和离群点不敏感。但是，对于密度变化较小的数据集，Dbscan 表现欠佳；另外，由于高纬数据的密度定义更加困难，采用 Dbscan 的效果也不理想。

4．模型聚类

从概率分布的角度考虑问题，并依据概率的大小判定类别，这种方式称为模型聚类。假定不同类别的数据组成的数据集具有不同的概率公式，只要获取该概率公式，即可判定输入数据的类别。GMM（高斯混合模型）是一种典型的模型聚类，我们举例说明。假设统计某高校男女的身高，得到数据 *X* = (173.3, 175.1, 165.2 , ···, 180.2)为身高记录，*Y* = {"boy", "boy", "girl", ···, "boy"}为对应的性别数据；男女的身高服从正态分布，要求解的是如果已知某同学的身高是 1.60 米，该同学最有可能是男生还是女生？

该问题的求解简单明了。首先，把男生和女生身高数据划分成两部分，由于其符合正态分布，因此可以分别求得男生身高与女生身高的概率公式。概率分布图形如图 4.14 所示。

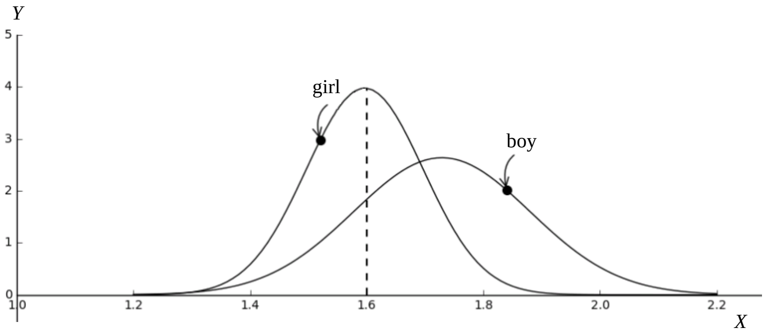


图 4.14　男女同学身高概率分布图

图 4.14 展示了男生与女生各自身高的概率密度函数曲线。同时，虚线部分标注了当该同学身高为 1.60 米时，其概率密度函数所对应的值。由图可知，该同学是女生的概率更高。

在此基础上提出另外一个问题：统计男生和女生的身高，依然得到数据 X = (173.3, 175.1, 165.2, ···, 180.2)的身高记录；但是，身高对应的性别记录 Y 丢失，即该数据集是一份无类别标注的数据集。那么，对于某位身高为 175.2 厘米的同学，如何判断其更可能是男生还是女生？显而易见，由于没有性别记录，因此无法求解男生和女生各自身高的正态分布概率密度公式，也就无法对比概率。

GMM 有效使用了 EM 的思想，迭代求解出各类别的多元高斯分布概率密度公式，进而实现了解决以上问题的模型聚类。该计算过程如下。

① 已知混合数据集由 K 类混合而成，每类皆服从多元高斯分布。
② 利用 EM 算法求解参数，进而得出每种类别下的数据集概率密度函数公式。
③ 依据概率公式求解每份样本属于某类的概率，并决定其类别归属。

第 2 步使用 EM 算法求解多元高斯分布的参数是整个建模的核心。由于 EM 算法涉及很多数学理论，并且许多资料中已有巨细靡遗的推导过程，因此，我们下面的讲解意在抛出问题，让读者对 EM 有初步的认知，并能以此为引子，进行更深入的学习。EM 算法的思想可以归纳为如下三步。

① 训练集 $X(x_1, x_2, x_3, \cdots, x_n)$是由 K 类数据组成的混合数据集，每种类别的数

据集均服从多元高斯分布。多元高斯分布模型的概率公式如下。

$$N(x;\mu,\Sigma) = \frac{1}{\sqrt{(2\pi)^n|\Sigma|}} \mathrm{e}^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)} \tag{4.12}$$

其中μ是均值，Σ是协方差矩阵，$|\Sigma|$是Σ的行列式，Σ^{-1}是Σ的逆。GMM 的目标就是求解每种类别的数据集对应的参数μ和Σ。

② 训练集样本数据为 x 的概率可以表述为

$$P(x|\mu,\Sigma) = \sum_{z=1}^{k} P(x|\mu,\Sigma)P(z|x,\mu,\Sigma) \tag{4.13}$$

其中 z 是一个隐藏变量，代表样本 x 所属的类别。

③ 根据概率公式 4.13，可以推导出当前数据集的发生概率如下。

$$\zeta(u,\Sigma) = \prod_{i}^{n} \sum_{z=1}^{k} P(x_i|u_i,\Sigma_i)P(z_i|x_i,u_i,\Sigma_i) \tag{4.14}$$

极大似然估计的思想是假定已发生的事情的概率最大。因此，使$\zeta(u,\Sigma)$达到最大参数，即是合理的参数。为了方便计算，对公式 4.13 两边取 log，得到如下公式：

$$L(\mu,\Sigma) = \log\zeta(u,\Sigma) = \log\left(\prod_{i}^{n}\sum_{z=1}^{k} P(x_i|u_i,\Sigma_i)P(z_i|x_i,u_i,\Sigma_i)\right)$$

$$= \sum_{i}^{n} \log \sum_{z=1}^{k} P(x_i|u_i,\Sigma_i)P(z_i|x_i,u_i,\Sigma_i) \tag{4.15}$$

然而，由于隐变量 z 的存在，出现了$\Sigma\log\Sigma$的形式，导致“梯度下降”“牛顿法”等常用求解极大似然估计的方法完全不可行。后面的逻辑回归章节，也用到了极大似然估计，读者可自行对比它与上述公式的区别。那么是否有可行的方法能对这样的公式进行参数求解？答案是 EM 算法。

4.1.3 回归

回归是指从历史数据中发现自变量与因变量的关系。线性回归、逻辑回归是两种常用的回归模型。线性回归是其他回归方式的基础；而逻辑回归广泛应用在

广告领域，也是所有回归算法中最重要的一种形式。除此之外，还有多项式回归、Lasso 回归、Ridge 等回归算法，它们都是从线性回归或逻辑回归演化而来的。

1．线性回归

线性回归是最简单的回归方式，是其他回归建模的基础。我们下面仍然根据示例逐步说明线性回归模型的建模理念及推导过程。

假设某门户网站的流量和广告带来的收益从长期来看存在简单的线性关系，当前需要根据流量数据的波动，预测营收的幅度，并且评估在网站运营带来更大流量的情况下，所带来的收益增长是否合理。

设每天的流量数据为 X 轴，广告收益为 Y 轴，取过去 N 天的数据，在二维坐标中画出简单的对应关系，如图 4.15 所示。

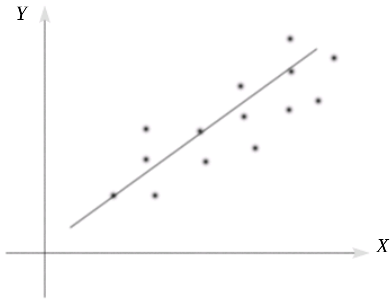


图 4.15　流量与收益示例图

线性回归就是要拟合出一条合理的直线，实现对收益 y 的预测。建模流程如下。

① 假设该拟合直线的数学公式为 $y = ax + b$；
② 图中各点的拟合值分别为 $y_i = ax_i + b$；
③ 拟合值与真实值的误差平方和 $\varepsilon = \sum_i^n (y_i - y'_i)^2 = \sum_i^n (ax_i + b - y'_i)^2$；
④ 求解使得 ε 最小的参数 a、b，从而得到线性回归的公式。

如上述步骤所述，先假定线性公式，定义损失函数，让损失函数最小的参数

是一种比较合理的取值。该例中选取“误差平方和最小”来逐步拟合未知参数的过程被称为最小二乘法；第 4 步中参数 a 和 b 的具体求解方法，一般使用随机梯度下降或者牛顿法。该例围绕一元线性回归讲述，同理，对于多元线性回归，只需把一元线性函数替换成$y=\theta_1 x_1+\ \theta_2 x_2,\cdots,\theta_n x_n$，参数拟合求解的过程并未变化。

2．逻辑回归

在介绍逻辑回归之前，先来认识一下 sigmod 函数，其数学公式为

$$g(z)=\frac{1}{1+\mathrm{e}^{-z}} \tag{4.16}$$

该公式对应的函数曲线如图 4.16 所示。

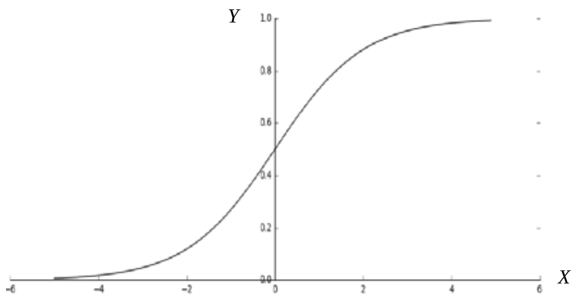


图 4.16　sigmod 函数示意图

由图 4.16 可知，sigmod 函数的几何图形是一条 S 形的曲线。y 的取值在[0, 1]之间；同时，x 越大 y 越接近 1，x 越小 y 越接近 0。这些性质都使 sigmod 在概率建模中得到广泛的应用。

如 *k*-means 一样，了解数据簇的形状对建模至关重要。图 4.17 中的两种图形分别代表了具有线性和非线性决策边界的簇。

如图 4.17(a)所示为具有线性决策边界的数据簇，图 4.17(b)是具有非线性决策边界的簇。

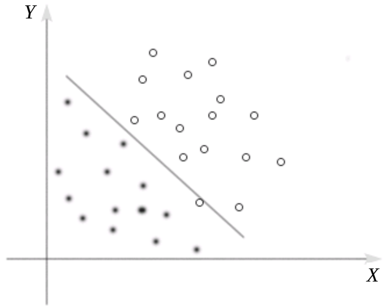


图 4.17(a)　线性分类与非线性分类示意图

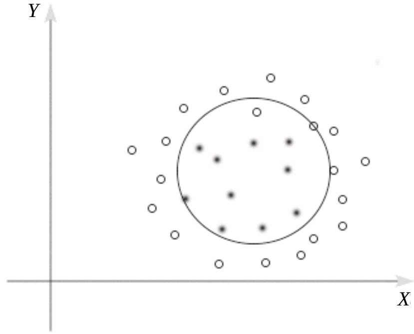


图 4.17(b)　线性分类与非线性分类示意图

对于前者，可以假设 $y = ax + b$ 为边界函数；对于后者，则假设 $(x+a)^2 + (y+b)^2 = 1$。逻辑回归就是把决策边界函数嵌套进 sigmod 函数，使得决策边界上与下，或者内与外之间的数据点具有大于 0.5 和小于 0.5 的概率值。相比于直接采用决策边界进行“是”或“否”的分类，概率值能方便地排序，在很多场景中更为实用。下面我们举例说明。

假定某智能医疗系统需要定期给出健康指导，它的某项功能是通过用户年龄、心率、血压、血脂等因子，判断用户是否需要一次全面的身体检查。

假设从历史体检记录中可以获取训练集(x_1, x_2, x_3, x_4, y)，x_i 代表各项检测指标，y=0 或者 y=1 代表是否需要体检。逻辑回归就是通过训练这样的数据集，获取模型参数，用于未来的体检预测。我们用形式化的方式描述其建模过程如下。

① 假设类别的决策边界函数 $\boldsymbol{y} = -\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}$，有{$y = 1, y = 0$}两类数据，则

$$P(y = 1\,|\boldsymbol{x}, \boldsymbol{\theta}) = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{1+\mathrm{e}^{-\theta^{\mathrm{T}}X}} \tag{4.17}$$

$$P(y = 0|\boldsymbol{x}, \boldsymbol{\theta}) = 1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}) \tag{4.18}$$

显而易见，如果能求得参数 $\boldsymbol{\theta}$，就可以求解特征向量 $\boldsymbol{X}$ 属于某类的概率值。

② 合并上述概率公式，$P(y\,|\,\boldsymbol{x}, \boldsymbol{\theta}) = (h_{\boldsymbol{\theta}}(\boldsymbol{x})^y)(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}))^{1-y}$　　(4.19)

③ 假定发生的事件概率最大，写出最大似然估计函数：

$$L(\boldsymbol{\theta}) = \prod_{i=1}^{n}(h_{\theta}(x_i)^{y_i})(1-h_{\theta}(x_i))^{1-y_i} \tag{4.20}$$

④ 为了计算方便，求其对数最大似然估计公式如下：

$$\zeta(\boldsymbol{\theta}) = \log\mathrm{L}(\boldsymbol{\theta}) = \log\prod_{i=1}^{n}(h_{\theta}(x_i)^{y_i})\left(1-h_{\theta}(x_i)\right)^{1-y_i}$$

$$= \sum_{i}^{n}\left(y^i log(h_{\theta}(x_i) + \left(1-y^i\right)\log\left(1-h_{\theta}(x_i)\right)\right) \tag{4.21}$$

⑤ 利用梯度下降或者牛顿方法，求解参数θ，使$\zeta(\boldsymbol{\theta})$最大化。

逻辑回归与线性回归是从不同的角度来解决问题：前者利用最大似然估计，即依据“已发生的事物概率最大”；后者采用最小二乘法，即“误差平方和最小”。同时，读者应该清楚地看到，两者都可以利用随机梯度下降或者牛顿方法来搜索解空间，从而获取拟合参数。逻辑回归从概率的角度给出“是”和“否”的概率，建模过程简单，易解释，因此在各领域得到广泛的应用。但是，概率建模往往伴随更多的要求，比如要求训练集必须具有一定的量，训练样本特征维度之间必须独立无关联等。

4.1.4 关联规则

关联规则旨在定义事物之间的关联性，并找出关联紧密的事物。关联挖掘常用于推荐领域，用以挖掘商品或者事务之间的关系，发现其中的关联性，依此制定推荐策略或者做出相关决策。

“啤酒与尿布”是和关联规则有关的一则广为人知的故事：某超市核查数据时发现，消费者在买啤酒时，总是会顺手买一些尿布。因此，该超市把啤酒和尿布的位置调近，从而提高了两者售卖的数量。如今，像亚马逊、淘宝等电商平台都有相关的推荐模块，毋庸置疑，关联规则挖掘是它们所应用的技术之一。

Apriori 和 FP-Tree 是关联规则挖掘中常用的两种经典算法，后者是针对前者在建模过程中存在的问题提出的一种改进算法。因此本节重点阐述 Apriori 的建模

过程。

“支持度”“置信度”“频繁项集”是关联规则挖掘中三个关键的概念，我们举例说明。假设有{牛奶、面包、苹果、鸡肉}四种食品，每位顾客每次购买的记录数据如表 4-11 所示。

表 4-11　购物篮记录

顾客 ID	购买商品
1	牛奶、苹果
2	牛奶、苹果、鸡肉
3	面包、苹果
4	牛奶、面包、苹果、鸡肉
5	牛奶、鸡肉
6	苹果

由表 4-11 可知，{牛奶、苹果}共同购买的记录是 3，牛奶的购买记录数是 4，总的购买记录是 6。因此，“买牛奶”→“买苹果”的置信度= 3 / 4 = 75%，支持度 = 3 / 6 = 50%。如果设定最小支持度是 50%，那么支持度大于等于 50%的项的组合称为频繁项集。

集合 I={牛奶、面包、苹果、鸡肉}，称作项目集合。每位顾客一次购买商品的集合 t 称为事物，$\{t_1, t_2, t_3, \cdots, t_n\}$称为事务集合。规则形式如：$X \rightarrow Y$，其中 X, Y 是 I 的真子集，并且 X 和 Y 的交集为空；X 称为前件，Y 称为后件。

“支持度”反映项集在全部事务集合中出现的次数，设定支持度阈值，可以屏蔽那些很少同时出现的项目组合，这些组合大多都是偶然出现且无意义的；“置信度”反映规则推理的可靠性，$X \rightarrow Y$ 的置信度越高，那么在出现 X 的场景中，Y 出现的概率越高。

Apriori 算法是关联规则挖掘领域最经典的算法，用于寻找“频繁项集”中满足一定置信度的关联规则。我们援引上例简要说明它的建模过程。

① 设定支持度≥50%的项集为频繁项集；置信度设为 50%。

② 先处理单个项集，统计数据如表 4-12 所示。

表 4-12　关联规则构建过程

项	计　　数	支持度
牛奶	4	4 / 6
面包	2	2 / 6
苹果	5	5 / 6
鸡肉	4	4 / 6

③ 根据“任何频繁项集的子集都是频繁项集”，可以删掉非频繁项集再进行下一步计算，如下：

项	计数	支持度
牛奶	4	4 / 6
面包	2	2 / 6
苹果	5	5 / 6
鸡肉	3	3 / 6

→

项	计数	支持度
牛奶、苹果	3	3 / 6
牛奶、鸡肉	3	3 / 6
~~苹果、鸡肉~~	~~2~~	~~2 / 6~~

↓

项	计数	支持度
~~牛奶、苹果、鸡肉~~	~~2~~	~~2 / 6~~

④ 得到{牛奶}、{苹果}、{鸡肉}、{牛奶、苹果}、{牛奶、鸡肉}五个频繁项集。计算{牛奶→苹果}、{牛奶→鸡肉}的置信度。此时，不需要再遍历数据集，只要根据上述算法过程中记录的支持度，即可求解置信度。{牛奶→苹果} ={牛奶，苹果}的支持度 / {牛奶}的支持度 = 3 /4 = 75%。因此，可以得到{牛奶→苹果}这条规则。

Apriori 算法逐层求解频繁项集，并在该过程中依据“任何频繁项集的子集都是频繁项集”这条推论剪枝；最终找到所有的频繁项集。它使用中间结果，计算所有满足置信度的规则。但是，该算法在求解频繁项集的过程中需要遍历数据集，这种遍历行为极大增加了整个建模过程的运算量，因此 FP-Growth 算法对此进行了完善，有兴趣的读者可以自行研究。

4.2　数据挖掘的重要概念

4.2.1　数据预处理

（1）距离的选择

数据挖掘中“距离”代表数据之间相似与相异的程度。在聚类、分类和其他相关建模中，距离的选择都十分重要，不同的场景应选择不同的距离来度量。除了常用的欧式距离，还有曼哈顿距离、余弦相似度、Hamming 距离等多种距离。

① 欧氏距离是各类算法中最常用的距离。通常所说的平面或者空间中两个点之间的距离，默认的就是指欧式距离。它也是多维空间中点与点之间的距离度量，它的数学公式定义如下。

$$d = \sqrt[2]{\sum_{i=1}^{n}(x_{1i} - x_{2i})^2} \tag{4.22}$$

其中 n 代表 n 维向量，a =（$x_{11}, x_{12}, x_{13}, \cdots, x_{1n}$）和 b =（$x_{21}, x_{22}, x_{23}, \cdots, x_{2n}$）是 n 维向量中的两个点。

② 曼哈顿距离又称为“L1-距离”或者城市区块距离，由十九世纪的赫尔曼・闵可夫斯基所创。曼哈顿距离的直观展示如图 4.18 所示。

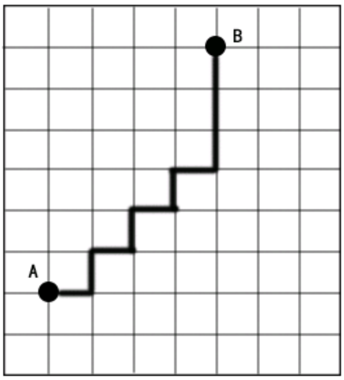


图 4.18　曼哈顿距离示意图

A、B 是二维平面坐标系中的两点，图中折线就是曼哈顿距离，它的数学定义

公式如下。

$$d = \sum_i^n |x_{1i} - x_{2i}| \tag{4.23}$$

在 *k*-means 算法中，为了克服异常点的影响，引入了 *k*-medians，它用各点的中位数作为质心，数据点与质心的距离采用曼哈顿距离。

③ 余弦相似度，通过测量两个向量的夹角的余弦值来度量它们之间的相似性，常用于文本相似度的计算中。它的数学定义公式如下。

$$d = \cos(\boldsymbol{\theta}) = \frac{\vec{a}\,\vec{b}}{||\vec{a}||\ ||\vec{b}||} = \frac{\sum_i^n x_{1i}\,.\,x_{2i}}{\sqrt{\sum_i^n x_{1i}{}^2} \times \sqrt{\sum_i^n x_{2i}{}^2}} \tag{4.24}$$

余弦相似度确定了两个向量之间的方向是否一致。当向量之间有相同的指向时，其值为 1，当向量之间夹角为 90°时，其值为 0。文本可以用词频组成的向量表示，用余弦相似度度量文本之间的距离，不受各维向量值幅度大小的影响。

④ “长度相同”的两个字符串采用 Hamming 距离度量。它的距离定义：在两个字符串相互对应的位置所出现的不同字符的个数。比如“abcef”和“abcmn”的 Hamming 距离是 2；“11001”和“10110”的 Hanming 距离是 4。

⑤ 和 Hamming 距离相比，Levenshtein 距离不要求对比的字符串具有相同的长度。它的距离定义：通过替换、插入、删除操作，把一个字符串转换为另一个字符串所需要的最小编辑次数。比如“abc”和“ab”的 Levenshtein 距离是 1；“1101”和“110111”的 Levenshtein 距离是 2。

⑥ SMC 系数与 Jaccard 系数度量仅包含二元属性的对象的相似度。假设有 *x*=(1, 0, 0, 0, 0, 0)和 *y*=(0, 1, 0, 0, 0, 0)，我们定义如下四个变量：

$f_{00} = x$取 0 并且y取 0 的属性个数

$f_{01} = x$取 0 并且y取 1 的属性个数

$f_{10} = x$取 1 并且y取 0 的属性个数

$f_{11} = x$取 1 并且y取 1 的属性个数

则 SMC 与 Jaccard 系数的数学公式分别如下：

$$\text{SMC} = \frac{f_{11} + f_{00}}{f_{01} + f_{10} + f_{11} + f_{00}} \tag{4.25}$$

$$\text{Jaccard} = \frac{f_{11}}{f_{01}+f_{10}+f_{11}} \tag{4.26}$$

由上述公式可知，SMC 系数和 Jaccard 系数的公式只是分子和分母中去掉了 f_{00}，因为某些场景下需要考虑 0 值过多的问题，所以采用 Jaccard 系数。比如，网上商场中卖掉某种商品记录为 1，否则记录为 0，那么 0 的个数将远大于 1，此时选用 Jaccard 系数更合理。

以上几种距离作为度量的方式，经常运用在各类算法中。除此之外，还有马氏距离、DTW 距离等。只有依据特定的场景选择合适的距离，才能构建更加完善的模型，达到更好的数据挖掘效果。

（2）数据归一化

数据归一化的目的是消除量纲的影响，把数据转换到同一量级，从而使其具有可比性。比如使用{年龄，体重，身高}说明两个人在感官上的相似度。显而易见，由于三个维度的量纲不同，不能直接度量距离，归一化正是为了解决类似这样的问题。

数据归一化的方法包括：Min-Max、Z-score、log 函数转换，atan 函数转换等方式。其中 Min-Max 和 Z-score 是比较常用的两种方法。

① Min-Max 方法

该方法对原始数据做线性变换，使结果映射到[0 - 1]之间。数学公式定义如下。

$$x' = \frac{x-\min}{\max-\min} \tag{4.27}$$

Max 是训练集中所有特征向量在 x 所在维度的最大值；同理 Min 是其最小值。比如训练样本有{(1, 9, 9), (2, 8 ,10), (5, 10, 8)}，对特征向量(1, 9, 9)进行 Min-Max 归一化，$\left(\frac{1-1}{5-1}, \frac{9-8}{10-8}, \frac{9-8}{10-8}\right) = (0, \frac{1}{2}, \frac{1}{2})$。应用此方法需要注意当有新的样本加入时，需要重新归一化各特征向量。

② Z-score 方法

该方法利用均值和标准差对数据实施归一化。数学公式定义如下。

$$x' = \frac{x-\mu}{\delta} \tag{4.28}$$

其中μ、δ是“母体”的平均值与标准差，并非取决于训练样本。但是除非“母体”符合一定的分布，不然无法得知其均值与标准差。因此，一般通过采样获取随机样本，使用样本的均值和标准差来进行 Z-score 归一化。同时，如果某维度的数据符合正态分布，那么变换之后的数据服从标准正态分布。

（3）缺失值处理

在现实的数据训练集中，各种缺失值的存在会影响建模过程中的计算，或者导致最终结果的偏差。一般要视场景的差异，对缺失值进行“删除”或者“填补”处理。比如，当存在维度缺失的样本占比过小，或者样本缺失的维度过多时，直接删除该样本是更好的选择；然而，当维度缺失少，且该类样本又占有一定的比例时，采用一些方法对其填补则更合适。一些常用的缺失值填充方法包括拉格朗日插值法、牛顿插值法、均值法、最近填补法。

牛顿插值法与拉格朗日插值法都是多项式插值法。它们的共同思想是给出一个恰好可以穿过二维平面若干点的多项式函数，即如果已知若干点(x_i, y_i)，求解其对应的多项式函数，进而获取未知点$(x_i, ?)$的数值。用图像描述如图 4.19 所示。

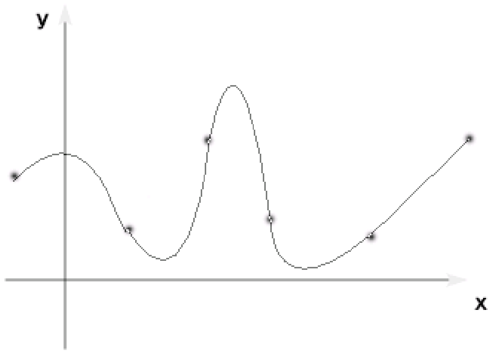


图 4.19　多项式插值法示意图

多项式插值法对已知的 N 个点，构造如图 4.19 所示的多项式函数，该函数的曲线能够平滑地贯穿已知的 N 个点。“均值法”和“最近填补”是较为简单且十分常用的方法。顾名思义，所谓均值法就是采用缺失值所在维度的均值填补该缺失值；而最近填补则是使用数据中该缺失值的前一项或者后一项的值来填充，我们用如表 4-13 的数据来说明。

表 4-13 缺失值数据集

维度 A	维度 B	…	维度 N
12	0		312
31	Nan		31
231	1.20		192
23	2.51		520
77	3.03		64

如表 4-13 所示，Nan 代表缺失值。运用均值法填充该缺失值，均值等于(0 + 1.2 + 2.5 + 3.0) / 4 = 1.68，所以采用 1.68 对其填充；运用最近填补法填充该缺失值，Nan 可以选取 1.20 作为其数值估计。

4.2.2 评估与验证

（1）AUC 和 ROC

在构建好模型后，需要量化评估该模型的优劣。量化评估的一些常用的指标包括分类错误率、分类准确率，预测误差范围等。AUC 和 ROC 是评估“分类”算法最常用的指标，在 CTR 预估以及其他建模评估中几乎随处可见。

首先，我们先抛出一个问题来说明为什么要引入 AUC 和 ROC。假设待分类的数据集有 100 份样本数据，其中 A 类样本 10 份、B 类样本 90 份。在这样的数据集中，如果有模型把所有的样本都归为 B 类，其准确率也有 90%，显然这是不合理的。因此，在数据集正负样本数量差别较大时，需要有一种新的评价指标。

针对二分类问题，假设有正类（Positive）和负类（Negative），实际的分类结果有以下四种情况。

① 样本为正类，且被划分到正类，称其为真正类（True Positive），简称 TP；
② 样本为正类，且被划分到负类，称其为假负类（False Negative），简称 FN；
③ 样本为负类，且被划分到正类，称其为假正类（False Positive），简称 FP；
④ 样本为负类，且被划分到负类，称其为真负类（True Negative），简称 TN。

由表 4-14 可得出以下定义。

① 真正类率（True Positive Rate）TPR : TP / (TP + FN)；
② 假正类率（False Positive Rate）FPR: FP / (FP + TN)；
③ 真负类率（True Negative Rate）TNR: TN / (FP + TN)；
④ 假负类率（False Negative Rate）FNR:FP / (FP + TN)；

表 4-14　分类结果类别

		真实值		总数
		P	N	
预测值	P	TP	FP	TP + FP
	N	FN	TN	FN + TN
总数		TP + FN	FP + TN	

以逻辑回归为例，ROC 是这样的一条曲线：以 FPR 为横轴、以 TPR 为纵轴，并不断调整类别判定的阈值，得到不同的(FPR, TPR)，这些点最终在二维平面上绘制成一条曲线。假设该曲线如图 4.20 所示。

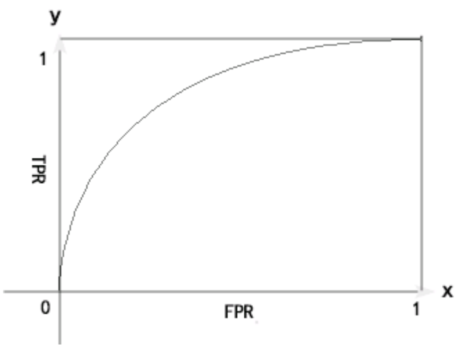


图 4.20　ROC 示意图

由定义可以，TPR = 1 且 FPR = 0 代表最好的结果。假设将模型 A 与模型 B

相比，模型 A 的 ROC 曲线整体更加趋近(0,1)这个点，那么该模型较好。为了量化这种趋势，我们引入 AUC 的概念（即曲线下方的面积），该值大于 0，小于 1。因此，AUC 的值越大代表该模型越好。同时，ROC 和 AUC 不会因正负样本占比的差异而有大的波动。

（2）交叉验证

在数据挖掘建模的同时，我们需要设计验证方案以说明构建模型的有效性。交叉验证（Cross-Validation）是模型验证的一种常用方式，它有三种具体的实现策略：普通交叉验证、*K* 折叠交叉验证、留一交叉验证。

普通交叉验证把数据分成两组：训练数据集和测试数据集。用训练数据集训练出模型后，再在测试数据集上做测试，进一步评估模型的分类准确率等相关指标。

K 折叠交叉验证把数据分为 *K* 组，其中 *K*-1 组作为训练数据集，用于模型的构建，剩余的 1 组数据作为测试集，用于模型优劣的度量。同时，不断变更 *K*-1 的组合，最终得到 *K* 份验证结果，取其均值作为最终的衡量指标。

留一交叉验证是 *K* 折叠交叉验证的一个特例，即如果有 *N* 个样本，则把数据分为 *N* 组，每次都留一个样本作为测试数据集。

4.2.3　Bagging 与 Adaboost

当弱分类器的分类效果不理想时，我们要考虑是否可以组合多个弱分类器以提升分类效果？事实证明，确实存在一些方法可以使组合后的弱分类器具有惊人的效果。本节介绍的 Bagging 和 Adaboost 正是两种典型的弱分类器聚合方式。

Bagging 对训练数据集采用有放回的采样。每一次采样都可以获得一个弱分类模型，所以，经过 *K* 次采样建模，最终获取 *K* 个分类模型；并由 *K* 个模型的分类样本投票表决，最终决定其类别。该过程如图 4.21 所示。

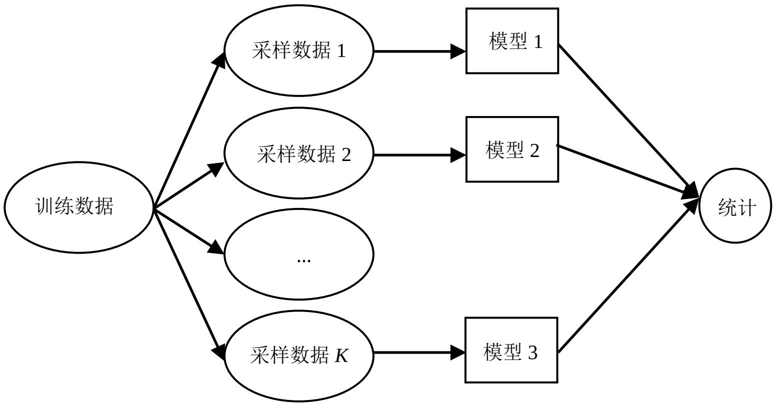


图 4.21　Bagging 模型聚合示意图

图 4.21 清晰展示了 Bagging 模型的聚合过程：根据采样数据训练子分类模型，并以投票的方式决定待分类样本的类别。

Adaboost 是“Adaptive Boosting”的英文缩写，由 Yoav Freund 和 Robert Schapire 提出。Bagging 投票表决的方式忽略了各模型的权重，与 Bagging 不同，Adaboost 的建模思想是通过对模型加权，将多个弱分类器聚合成强分类器。它的算法流程如下。

① 训练分类器 A，设初始样本权值 $w_0 = 1 / N$，统计该模型分类误差 $\varepsilon = w_1 + w_2 + w_3 + \cdots\ w_n$。依据公式

$$\frac{1}{2}\ln\left(\frac{1-\varepsilon}{\varepsilon}\right) \tag{4.29}$$

求解该分类器对应的权值。

② 依据公式

$$w_n = w_{n-1} \times \left(\frac{1-\varepsilon}{\varepsilon}\right) \tag{4.30}$$

调大被分类器 A 错判的样本的权值。

③ 以同样的方式训练分类器 B，统计分类误差ε，进而求解模型权值。

④ 依次类推，完成 N 个分类器的权值计算。

⑤ 各模型依据权值对待分类数据投票。

用图形化示例来说明上述过程如图 4.22 所示。

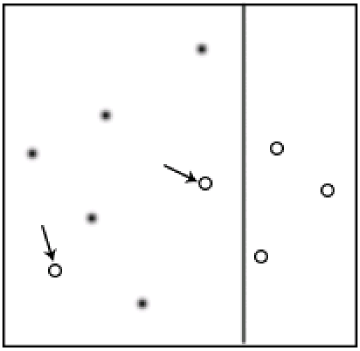


图 4.22(a)训练器 A

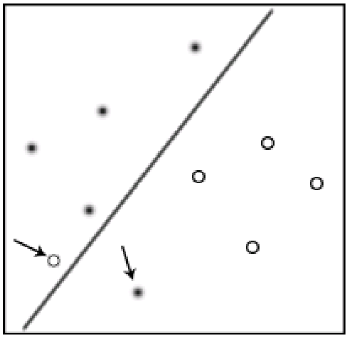


图 4.22(b)训练器 B

如图 4.22(a)所示，竖线代表分类边界，箭头所指的空心圆代表被错误分类的点。

① 训练器 A，初始各点权值 $w_1 = 1 / 10 = 0.1$，错误率$\varepsilon = 0.1 + 0.1 = 0.2$。

因此，模型 A 的权重 $= \frac{1}{2}\ln\frac{1-\varepsilon}{\varepsilon} = \frac{1}{2}\ln\frac{1-0.2}{0.2} = 0.693$；

② 更新被错误分类的点的权重，箭头所指的点的权重 $w_2 = 0.1 \times (1 - 0.2) / 0.2 = 0.4$，当前权值之和等于$(0.1 \times 8 + 0.4 \times 2) = 1.6$；

③ 训练器 B，错误率$\varepsilon = 0.4 + 0.1 / 1.6 = 0.463$，模型 B 的权重$= \frac{1}{2}\ln\frac{1-0.463}{0.463} = 0.160$；

④ 依次训练其他模型，迭代计算各模型权重，根据权值进行分类投票。

Bagging 和 Adaboost 聚合模型极大提升了分类效果，几乎是建模的必用技能。大多数情况下，只要分类器的效果略高于随机选取，都可以使用 Bagging 或者 Adaboost 方法将其组合成强分类器。像 GBDT、随机森林等聚合模型，已在工业领域得到广泛的应用，并且表现不俗。

4.2.4 梯度下降与牛顿法

梯度下降是一种用来找到函数局部极小值的一阶最优化算法。牛顿法则是在实数域和复数域求解方程近似解的方法。在数据挖掘以及深度学习的建模过程中，经常使用梯度下降和牛顿法来求解参数。

线性回归、逻辑回归、模型聚类等建模都具有相似的流程：

① 假设模型函数$h_{\boldsymbol{\theta}}(\boldsymbol{x})$；
② 利用最小二乘或者极大似然的思想，构造损失函数 $J(\boldsymbol{\theta})$；
③ 最小化损失函数，求解参数$\boldsymbol{\theta}$。

梯度下降和牛顿法正是第 3 步中求解$\boldsymbol{\theta}$的两种方式。下面我们以线性回归为例，简述这两种方法。

假设定义线性模型的边界函数为

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 \tag{4.31}$$

转化成一种更为通用的表达方式为

$$h(\boldsymbol{x}) = \sum_{j=0}^{n} \theta_j x_j = \theta^{\mathbf{T}} x \tag{4.32}$$

其中 n 代表参数的个数。依据最小二乘法的思想构造损失函数如下。

$$J(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{m}(h_{\boldsymbol{\theta}}(x_i) - y_i)^2 \tag{4.33}$$

求解参数$\boldsymbol{\theta}$，使得损失函数$J(\boldsymbol{\theta})$最大，其中 m 代表样本数。

（1）梯度下降

梯度下降法是一种遍历搜索解空间，找到损失函数局部极小值的方法。依据 AndrewNg 的讲义，通常如图 4.32 所示的二次函数的解空间是“碗状”形态。

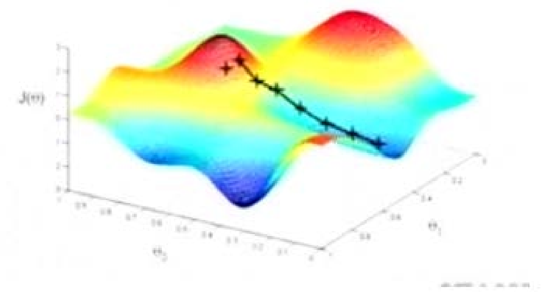


图 4.23 随机梯度下降解空间

随机取一个点，沿着梯度下降的方向修正参数$\boldsymbol{\theta}$的值。这个过程类似登山，从一点出发，朝着当前坡度最大的方向，向上或者向下走，最终到达当前所能看到的最高的山顶或者谷底。需要注意：只有当函数为凸函数时，梯度下降的解才是全局最优解。

参数的梯度就是$J(\boldsymbol{\theta})$对该参数求导的结果，因此，各个参数在逐步迭代的过程中，按如下公式调整。

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) = \theta_j - \alpha \sum_i^m (y^{(i)} - (h_{\theta}(x^{(i)}))x_j^{(i)} \quad (4.34)$$

其中α代表每次沿梯度方向延长的步长。依据上述公式不断迭代更新参数，直到$J(\boldsymbol{\theta})$的值不再有显著的变化。

当样本数据 m 过多时，上述训练过程的每次迭代更新参数$\boldsymbol{\theta}$，都需要计算所有的样本数据，这对计算能力是一个巨大的挑战。因此，有人提出了“随机”梯度下降的方法。每次迭代只对一个样本数据求导，更新参数$\boldsymbol{\theta}$。

$$\text{Loop}\{\text{for i =1 to m } \{\theta_j = \theta_j - \alpha(y^{(i)} - (h_{\theta}(x^{(i)}))x_j^{(i)}\}\} \quad (4.35)$$

每次迭代都忽略其他样本，而让单个样本趋于最小，损失函数曲折地向极值靠拢。

（2）牛顿法

逻辑回归建模中损失函数$J(\theta)$存在一阶偏导数，以及大于 0 的二阶偏导数，求解使得$J(\theta)$的一阶偏导数等于 0 的参数θ,则该参数可以让损失函数取得极小值。与随机梯度相同，只有函数是凸函数时，该极小值才是最小值。

牛顿法正是在场景中求解参数的θ的方法。使用$f(x)$二元函数简述牛顿法的基本过程，进而推广到类似$J(\theta)$的 N 元函数中使用。

首先，随机选择 x_0，求解 $f(x_0)$及其斜率$f'(x_0)$。计算以$f'(x_0)$为斜率，穿过点$(x_0,f(x_0))$的直线与 X 轴的交点 x_1，即依据如下公式求得。

$$(x_1 - x_0)f'(x_0) + f(x_0) = 0 \tag{4.36}$$

推导出

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \tag{4.37}$$

使用点$(x_1, f(x_1))$，执行下一轮计算，迭代公式抽象成如下形式。

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{4.38}$$

根据上述公式，不断迭代更新，直到$f'(x)$ 趋于 0。该过程如图 4.24 所示。

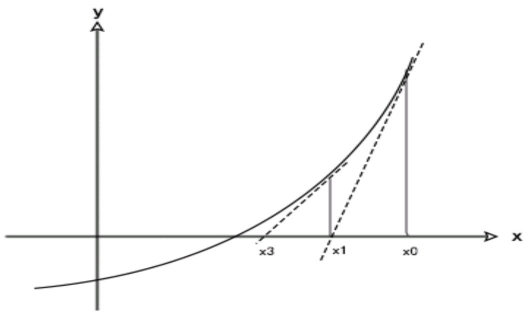


图 4.24 牛顿法迭代过程

如图 4.24 所示，经过 N 轮的迭代计算，最终可以获取使 $f(x)$无限接近于 0 的

x。使用牛顿法求解 $J(\theta)$，N 元函数的具体过程比上述复杂，但思想是完全一致的。相比于随机梯度下降法，牛顿法是二阶收敛，因此，收敛速度更快；牛顿法在迭代求解的过程中，涉及对 Hessian 矩阵的计算，不适合特征值种类过多的样本数据。

4.3　实例讲解

4.3.1　信用卡欺诈监测

（1）背景说明

数据来自 Kaggle 平台提供的公开数据集，记录了欧洲信用卡机构对 2013 年 9 月的某两天 284,807 次信用卡的消费记录。其中 492 次已被标记为欺诈交易，占比约 0.172%，是一份极度倾斜的数据集[14]。

该数据集中每份样本具有 30 个特征维度。出于保密考虑，除了“Time”和“Amount”两项以外，其余特征值都是经过 PCA 降维处理后的数值型特征，且已缺失背景信息。“Time”特征说明每个交易事务和该数据集中第一次事务之间的时间间隔，单位为秒；“Amount”为该次交易的数额；“Class”代表是否为欺诈交易——1 代表是，0 代表否。

此次分析的目的是希望建立有效的模型，对测试数据集的交易记录进行欺诈评估，以有效地区分正常交易与欺诈交易。下面我们逐步阐述该建模过程。

（2）数据建模

```
import numpy as np
import pand as as pd
import matplotlib.pyplot as plt
from sklearn.cross_validation import train_test_split
from sklearn import tree
from IPython.display import Image

#第一步：数据探究
data = pd.read_csv('D:\dateset\HR_comma_sep.csv')
#①数据样本
```

```
data.head()
#②样本数
classStay = data[data.left == 0]
classLeave =data[data.left == 1]
#③数据倾斜程度
print "%s%s"%("CLass == 0 的样本数：", classStay.shape[0])
print "%s%s"%("CLass == 1 的样本数：", classLeave.shape[0])
#④查看缺失值
data.info()
```

首先，依据相关经验和具体场景，从多个维度探究数据。这个过程并不存在约定俗成的方法，我们一般会从数据的格式、数据的样本数、倾斜度等维度入手。上述代码的输出结果如下。

...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0

```
数据总样本数： 284807
Class == 0 的样本数：284315
Class == 1 的样本数：492
Datacolumns (total 31 columns):
Time      284807 non-nullfloat64
V1        284807 non-nullfloat64
V2        284807 non-nullfloat64
V3        284807 non-nullfloat64
V4        284807 non-nullfloat64
V5        284807 non-nullfloat64
V6        284807 non-nullfloat64
V7        284807 non-nullfloat64
V8        284807 non-nullfloat64
V9        284807 non-nullfloat64
V10       284807 non-nullfloat64
V11       284807 non-nullfloat64
V12       284807 non-nullfloat64
V13       284807 non-nullfloat64
V14       284807 non-nullfloat64
V15       284807 non-nullfloat64
V16       284807 non-nullfloat64
```

```
V17      284807 non-nullfloat64
V18      284807 non-nullfloat64
V19      284807 non-nullfloat64
V20      284807 non-nullfloat64
V21      284807 non-nullfloat64
V22      284807 non-nullfloat64
V23      284807 non-nullfloat64
V24      284807 non-nullfloat64
V25      284807 non-nullfloat64
V26      284807 non-nullfloat64
V27      284807 non-nullfloat64
V28      284807 non-nullfloat64
Amount    284807 non-nullfloat64
Class     284807 non-nullint64
dtypes: float64(30), int64(1)
memoryusage: 67.4 MB
```

数据大小为 67.4MB，且不存在缺失值；Class 类别的倾斜较为严重。因此，在预处理中不需要特别处理缺失值；但需要对数据进行采样，以均衡正负样本的数量。同时，对“Amount”做归一化，去除量纲的影响。

```
#第二步：数据预处理
#①数据归一化
data['normAmount'] = StandardScaler().fit_transform(data['Amount'].
reshape(-1, 1))
#②去除无用特征
data = data.drop(['Time','Amount'],axis=1)
data.head()
#③数据采样
classNormalIndex = classNormal.index
fraudNum = classFraud.shape[0]
SampleNormalIndex = np.random.choice(classNormalIndex, fraudNum,
replace=False);
underSampleIndex  =np.concatenate([SampleNormalIndex, classFraud.
index])
#underSampleIndex =np.random.choice(underSampleIndex,
len(underSampleIndex), replace=False);
underSampleData = data.iloc[underSampleIndex,:]
sampleX = underSampleData.ix[:, underSampleData.columns != 'Class']
sampleY = underSampleData.ix[:, underSampleData.columns == 'Class']
```

从预处理之后的数据集中划分训练数据和验证数据，分别用于模型的训练和后续的验证评估。

```
    #第三步：划分训练数据集和验证数据集
    trainX, testX, trainY, testY = train_test_split(sampleX, sampleY,
test_size = 0.2, random_state = 0)
```

在训练数据集上训练逻辑回归模型，并使用 K-Fold 交叉验证，选取最优参数，其中 K 选取 5。

```
    #第四步: 数据建模与交叉验证参数
    #①交叉验证数据集
    fold = KFold(trainY.shape[0], 5, shuffle=False)
    cParams   = [0.01, 0.1, 1, 5, 10, 20, 50, 100]
    results = pd.DataFrame(index = range(len(cParams),2), columns =
['C','MeanScore'])
    results['C'] = cParams
    i = 0
    forcincParams:
    recallAccMean = []
    foriteration, indexinenumerate(fold,start=1):
    lr = LogisticRegression(C = c, penalty = 'l1')
    lr.fit(trainX.iloc[index[0],:],
trainY.iloc[index[0],:].values.ravel())
    preY = lr.predict(trainX.iloc[index[1], :].values)
    recAccSco = recall_score(trainY.iloc[index[1], :].values, preY)
    recallAccMean.append(recAccSco)
    results.ix[i, 'MeanScore'] = np.mean(recallAccMean)
    i += 1

    reasonableC = results.loc[results['MeanScore'].idxmax()]['C']

    printresults
    printreasonableC
```

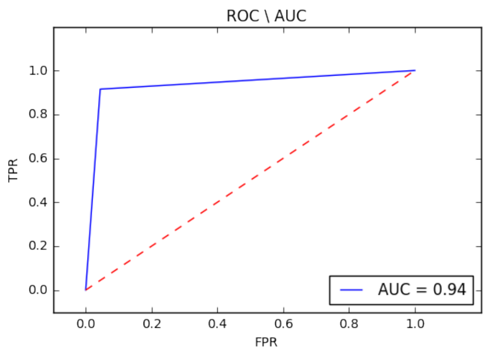
使用交叉验证的方式，选取使得误差均值最小的系数 C 作为最终的参数。上述程序的输出结果如下。

```
    CMeanScore
    0   0.01  0.928839
    1   0.10  0.899656
    2   1.00  0.911252
    3   5.00  0.913752
    4  10.00  0.918728
    5  20.00  0.918728
```

```
6   50.00  0.923273
7  100.00  0.923273
0.01
```

因此，选取 *C*=0.01 作为最终的模型参数。在测试集中验证模型，并依据 ROC 和 AUC 的指标评估模型的优劣。

```
#第五步：模型评估
#①测试集预测
lr = LogisticRegression(C = reasonableC, penalty = 'l1')
lr.fit(trainX, trainY.values.ravel())
preY = lr.predict(testX.values)
#②混淆矩阵
cnfMatrix = confusion_matrix(testY, preY)
printcnfMatrix
#③ROC 和 AUC 曲线
preYScore = lr.fit(trainX, trainY.values.ravel()).predict(testX.
values)
fpr, tpr, thresholds = roc_curve(testY.values.ravel(), preYScore)
roc_auc = auc(fpr, tpr)
plt.title('ROC \ AUC')
plt.plot(fpr, tpr, 'b',label='AUC = %0.2f'% roc_auc)
plt.legend(loc='lowerright')
plt.plot([0,1],[0,1],'r--')
plt.xlim([-0.1,1.2])
plt.ylim([-0.1,1.2])
plt.ylabel('TPR')
plt.xlabel('FPR')
plt.show()
 [[88  3]
 [10 96]]
```



根据程序输出的混淆矩阵，13 个样本数据被误判，占比约 6.60%；根据程序输出的 ROC\AUC 图形来看，虚线是以 50%的概率猜测分类结果时得到的 ROC 曲线图，其 AUC=0.5；实线是该模型所对应的 ROC 曲线，整体偏向于左上角，且 AUC = 0.94。因此，该逻辑回归建模整体较为理想。

4.3.2 员工离职预判

（1）背景说明

本数据集由 Twitter 提供，是一份伪造的数据集，一共有 14,999 份数据，并具有如下特征维度[15]。

① Employee satisfaction level
② Last evaluation
③ Number of projects
④ Average monthly hours
⑤ Time spent at the company
⑥ Whether they have had a work accident
⑦ Whether they have had a promotion in the last 5 years
⑧ Department
⑨ Salary
⑩ Whether the employee has left

该数据集旨在为人力资源部门研究员工离职的情况提供参考，希望通过建模的方式，在跳槽高峰期对员工是否离职做出合理的预判。该建模过程如下。

（2）程序建模

```
import numpy as np
import pandas as pd
import pyplot as plt
from sklearn.cross_validation import train_test_split
from sklearn import tree
```

```
fromIPython.displayimportImage

#第一步：数据探究
data = pd.read_csv('D:\dateset\HR_comma_sep.csv')
#①数据样本
data.head()
#②样本数
classStay  = data[data.left == 0]
classLeave  =data[data.left == 1]
#③数据倾斜程度
print "%s%s"%("CLass == 0 的样本数：", classStay.shape[0])
print "%s%s"%("CLass == 1 的样本数：", classLeave.shape[0])
#④查看缺失值
data.info()
```

同上节所述，我们首先从多个维度对数据进行直观的探究，再决定预处理所要做的工作。程序输出如下。

last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_last_5years	sales	salary
0.53	2	157	3	0	1	0	sales	low
0.86	5	262	6	0	1	0	sales	medium
0.88	7	272	4	0	1	0	sales	medium
0.87	5	223	5	0	1	0	sales	low
0.52	2	159	3	0	1	0	sales	low

```
Class == 0 的样本数：11428
Class == 1 的样本数：35715
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14999 entries, 0 to 14998
Datacolumns (total 10 columns):
satisfaction_level      14999 non-nullfloat64
last_evaluation         14999 non-nullfloat64
number_project          14999 non-nullint64
average_montly_hours     14999 non-nullint64
time_spend_company      14999 non-nullint64
Work_accident           14999 non-nullint64
left                    14999 non-nullint64
promotion_last_5years    14999 non-nullint64
sales                   14999 non-nullobject
salary                  14999 non-nullobject
dtypes: float64(2), int64(6), object(2)
memoryusage: 1.0+ MB
```

由上可知，共有 14,999 份样本数据，10 个特征维度，共 1MB 多的数据。同

时“sales”和“salary”两个属性是离散属性，在预处理中使用{0,1,···,*n*}数字代替。

```
    #第二步：数据预处理
    data['sales'].replace(['sales', 'accounting', 'hr', 'technical',
'support', 'management',
    'IT', 'product_mng', 'marketing', 'RandD'], [0, 1, 2, 3, 4, 5, 6,
7, 8, 9], inplace = True)
    data['salary'].replace(['low',  'medium',  'high'],  [0,  1,  2],
inplace = True)
```

在预处理的过程中，为了适应决策树建模相关库函数的要求，用数值替换“sales”和“salary”的离散属值。

```
    #第三步：划分训练集和测试集
    X = data.ix[:, data.columns != "left"]
    Y = data.ix[:, data.columns == "left"]
    trainX, testX, trainY, testY = train_test_split(X, Y, test_size =
0.2, random_state = 0)
```

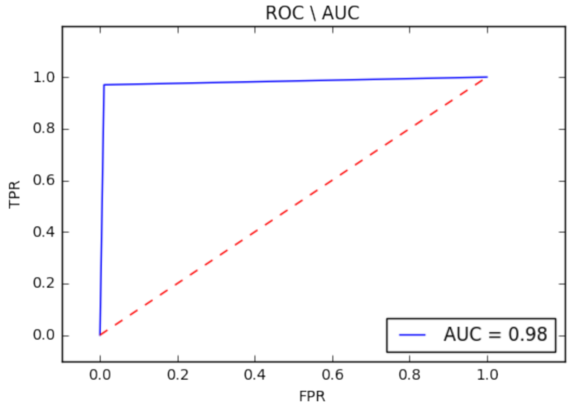
采用训练集占比 80%，测试集占比 20%的比例，划分数据集。划分后的数据，分别用于后续模型的训练和效果测试评估。

```
    #第四步：数据建模
    dt = tree.DecisionTreeClassifier(criterion='entropy',)
    dt.fit(trainX, trainY)
```

训练决策树分类模型，采用信息熵作为分枝分裂的考量指标。这里可以采用“K 交叉验证”对各项参数调优，上节已有相关代码，此处不再赘述。

```
    #第五步：模型评估
    trainScore = dt.score(trainX, trainY)
    print("Trainingscore: ",trainScore)
    testScore = dt.score(testX, testY)
    print("Testscore: ",testScore)
    preY = dt.fit(trainX, trainY).predict(testX);
    fpr, tpr, thresholds = roc_curve(testY.values.ravel(), preY)
    printfpr
    printtpr
    roc_auc = auc(fpr, tpr)
    plt.title('ROC \ AUC')
    plt.plot(fpr, tpr, 'b',label='AUC = %0.2f'% roc_auc)
```

```
plt.legend(loc='lowerright')
plt.plot([0,1],[0,1],'r--')
plt.xlim([-0.1,1.2])
plt.ylim([-0.1,1.2])
plt.ylabel('TPR')
plt.xlabel('FPR')
plt.show()
('Trainingscore: ', 1.0)
('Testscore: ', 0.98533333333333328)
[[2271   28]
 [  17  684]]
```



由输出可知模型在训练集的预测准确率为 100%，在测试集的预测准确率为 98.53%。ROC 与 AUC 的曲线也较为理想。因此，从当前评估来看该模型具有一定的预测效果。

上述两个示例，对数据挖掘建模的基本流程进行了简述。实际的工作中，模型的训练与评估需要做的工作不止于此。特征工程与预处理往往非常复杂，包括降维、去噪等操作；在一些场景中，合理的阈值更多是来源于经验；也有一些问题并不能通过套用公式和模型解决，而需要更有创意的思考。

5 深度学习

在第 4 章数据挖掘的学习中，读者可以清楚地认识到，训练数据集的特征维度往往来自业务经验，或需要经过多次尝试，最后由人工指定。这就使得机器学习仍然对“人工”具有很强的依赖。如何最大限度地降低这种依赖，并使机器具有更强的“智能”，正是深度学习所要解决的问题。

和数据挖掘（也被称为浅层学习）相比，深度学习是一种更加高级的数据处理技术。深度学习具有非监督或者半监督的学习模型，并且逐步弱化了对人工提取特征的依赖。绝大部分日常生活中的数据集是非线性的，深度学习在处理非线性的数据集时，比数据挖掘建模算法有更大的优势。这些特点使得深度学习技术具有更广泛的应用场景和更高的实用性。

1943 年，心理学家 Warren McCulloch 和数理逻辑学家 Walter Pitts 在论文中提出了人工神经网络的概念和人工神经元的数学公式，开创了深度学习的研究时代[17]。深度学习技术经历数十年的发展和完善，已由单一神经元演化到具有多种类型的复杂神经网络。“感知器”“编码器”“限制玻尔兹曼机”“深信度网络”和“卷积神经网络”的发现，是深度学习发展过程中的重要里程碑。

本章按照由易到难的顺序，逐步讲解各模型的建模理念与建模过程，并在最后的部分，基于 TensorFlow 框架，以“识别手写字体”和“让机器认识一只猫”两个实例，说明实践中构建神经网络的步骤。通过此章的内容，希望能抛砖引玉，让读者对深度学习有一定的理解，并能有的放矢地开展后续的研究。

5.1　场景分析与建模策略

5.1.1　感知机

Frank Rosenblatt 在 1957 年就职于 Cronell 航天实验室时发明了感知机[16]。它是由单个神经元组成的最简单的前馈神经网络，是深度学习的基础。感知机主要用来解决“线性数据”的二分类问题，网络结构如图 5.1 所示。

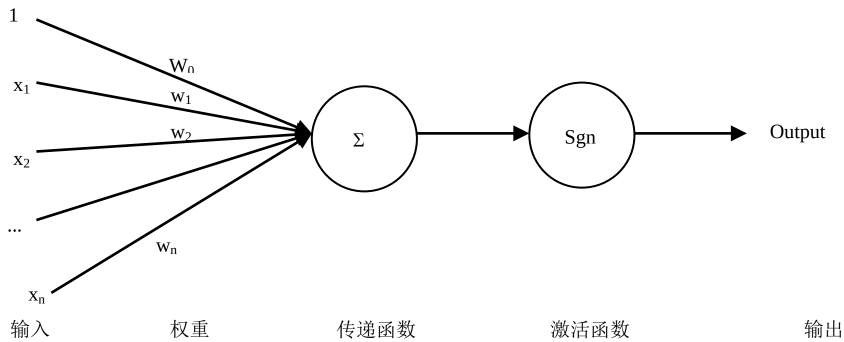


图 5.1　感知机示意图

如图 5.1 所示，$f(x) = w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n$，$f(x)$被称为传递函数。Sgn 为激活函数，其数学定义公式如下。

$$\mathrm{Sgn}(x) = \begin{cases} 1 \text{ if } x > 0 \\ 0 \text{ if } x = 0 \\ -1 \text{ if } x < 0 \end{cases} \tag{5.1}$$

可以看出，感知机的思路即是求解合理 $w_0,w_1,w_2,\cdots,w_n$，使“正”样本通过传递函数的计算，由激活函数输出 1；而“负”样本输出−1，实现二分类效果，这

和 logistic 回归极其相似。感知机的算法流程如下。

① 随机选取权值 w_i，构造传递函数$f(x)$。

② 进行样本分类，对误判的样本以如下的方式更新权值。

$$w_j = w_j + \Delta w_j \tag{5.2}$$

$$\Delta w_j = \varphi(\text{target}_j - \text{output}_j)x_j \tag{5.3}$$

其中target_j代表样本的真实分类，1 或者-1；output_j代表感知器的分类；x_j代表该维度上样本的值；φ代表迭代的步长，一般选取 0~1 之间的常数。

③ 重复步骤①和②，直到所有的模型可以正确判别所有的类别。

Frank Rosenblatt 已经从数学上证明当数据集线性可分时，感知机算法可以逐步收敛，直到所有的类别被正确的分类。在后面的章节中，我们还会介绍数据集线性不可分时的处理方法，在此之前，我们通过如下示例加深读者对感知机的认识。

假设有数据如表 5-1 所示，我们的任务是用这些数据构造感知机，并使数据可以被正确分类。

表 5-1　感知器训练数据

序　号	x_1	x_2	类　别
1	1	1	1
2	-2	3	-1
3	2	3	1

① 假设 $f(x) = 1 - x_1 + 3x_2$，即 $w_0 = 1, w_1 = -1, w_2 = 3$。设定迭代$\varphi = 0.5$。

② 样本序号 1 分类，output = sgn($f(x)$) = sgn(1 − 1 + 3) = 1，与 target 相同，权值不更新。

③ 样本序号 2 分类，output = sgn($f(x)$) = sgn(1 + 2 + 6) = 1，与 target 不同，更新权值。

$w_0 = 1 + 0.5 \times (-1 - 1) = 0$

$w_1 = -1 + 0.5 \times (-1 - 1) \times (-2) = 1$

$w_2 = 3 + 0.5 \times (-1 - 1) \times 3 = 0$

更新权值后的传递函数 $f(x) = 1 \times x_1 + 0 \times x_2 + 0 \times 1$。

④ 样本序号 3 分类，output = sgn($f(x)$) = sgn(2 + 3) = 1，与 target 相同，权值不更新。

⑤ 重新计算样本序号 1，感知机分类正确。

⑥ 重新计算样本序号 2，感知机分类正确。

⑦ 重新计算样本序号 3、感知机分类正确。

⑧ 迭代结束。

上述过程存在两个问题：一是当数据集线性不可分时，无法保证算法的收敛；二是即使数据集线性可分，运用感知器的迭代方式也往往无法获取最大的间隔平面，直接影响了算法的泛化效果。

Bernard Widrow 和 Tedd Hoff 提出了自适应线性神经元的概念。这个求解权值的过程与前面使用梯度下降法求解线性回归参数十分相似，但略有不同，它的思想如下。

① 随机选取权重 w_j，构造传递函数 $f(x)$。

② x_1、x_2... x_n 是函数的入参，数据集的分类-1 和 1 被看做目标函数的输出值。采用最小二乘法的思想来定义损失函数 $J(w) = \frac{1}{2}\sum_{i=1}^{n}(\mathrm{target}_i - f(x))^2$。

③ 样本分类，仍然只对误判的样本更新权值，但是$\Delta w_j = \frac{\partial J(w)}{\partial w_j}$，即运用随机梯度下降的思想，使得传递函数向步骤 2 中损失函数最小的方向更新权值。

④ 当样本的分类错误率达到一定阈值，或者权值的变化微小时，训练过程结束。

综上所述，当数据集线性可分时，可以保证感知机算法最终收敛并建立合适的模型；然而当数据集线性不可分时，该算法的收敛性无法得到保障。“异或”问题就是一个典型的感知机无法解决的线性不可分问题。

如图 5.2 所示，“圆圈”和“三角”分别代表两类数据。图 5.2(a)展示了线性可分的数据集；而图 5.2(b)中无法用一条直线分开两类数据，因此属于线性不可分的场景。同时，值得一提的是，第 4 章中数据挖掘的建模算法，通常会对数据集做线性可分的假设，虽然有时数据并非严格线性可分，也能通过这些挖掘算法

获得一定的效果。然而，现实问题中的数据集，绝大多数都是线性不可分的。是否有更为合适的方法，能对这类数据集做更优的处理？这就需要引入多层神经网络的概念。

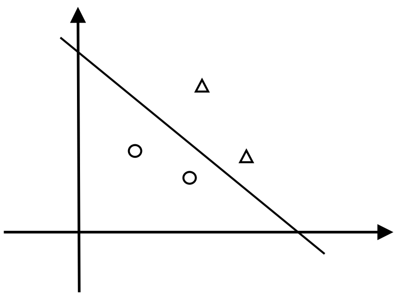


图 5.2(a)　线性可分的数据集

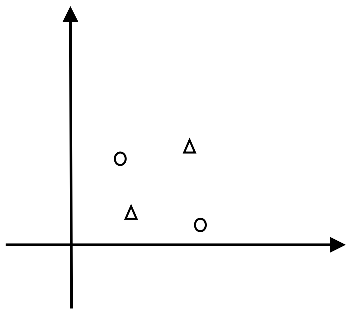


图 5.2(b)　线性不可分的数据集

图 5.3 是由两个隐层构建的多层神经网络的示意图。输入数据经过两层权值计算，由输出层输出。两层以及多层神经网络可以对线性不可分数据分类。

图 5.4 中的数据样本通过多层神经网络中隐层的处理,转换样本的表示方式。经过这样的变换，最终实现对线性以及非线性数据集的有效划分，这个流程和 SVM 的核函数有异曲同工之处。读者可在后面介绍的多种多层神经网络中，逐步体会深度学习处理数据的方式与数据挖掘方法的不同之处。

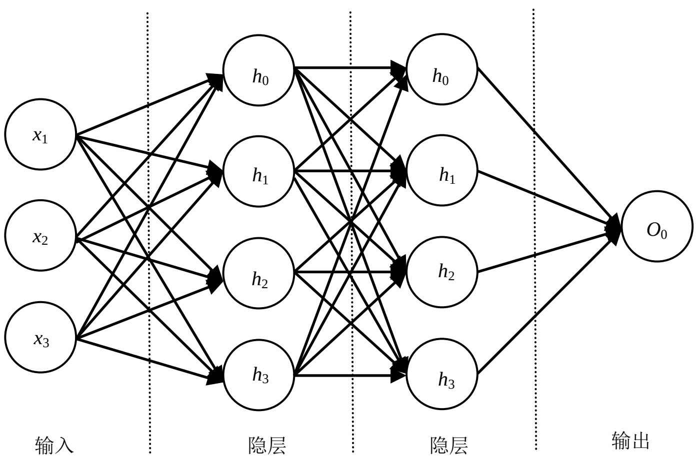


图 5.3　多层神经网络示意图

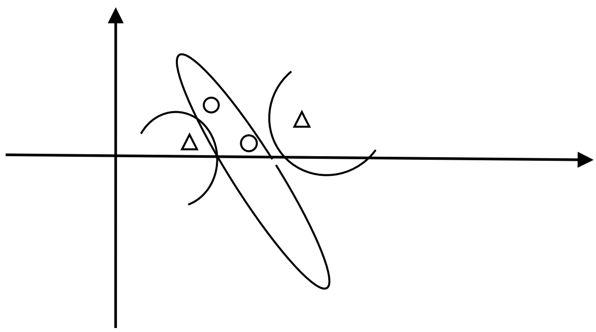


图 5.4　多层神经网络分类效果

5.1.2　自编码器

在信息通信领域，哈夫曼编码、香农编码常用于压缩信息，使得原有的信息量能用更少的二进制比特来表示。顾名思义，自编码器就是对自身编码，抽取最能代表数据本质的信息，去除无用数据的干扰，使用编码后的数据作为训练样本进行建模，能够得到更加优良的模型，对数据进行更加高效的学习建模。

比如，由{10011, 11, 11111, 001}四个样本组成的数据集，完全可以使用{00, 01, 10, 11}来编码。因此，从某种意义上来说，如果自编码器获取了输入数据的压缩表示，也就相当于进一步提取了数据所代表的信息的本质特征。

自编码器是一种无监督的学习算法。首先，对原始数据进行编码得到中间结果；再对中间结果进行解码；最后，训练编码与解码的参数，并通过对比原始数据与解码后数据的误差评估编码器的优劣。“中间数据”的维度均低于原始数据的维度，从而起到降维的作用。因此，“中间数据”可以看做是原始数据的压缩表示，或者说是原始数据更高阶特征的抽取。

如图 5.5 所示，数据 x_0~x_4 经过编码得到隐层数据 h_0~h_2，再次对数据解码，能够获得原始数据的恒等或近似值。隐层经过解码，得到输入数据的近似表示。因此，可以认为隐层和输入数据包含了同等的信息量；同时，隐层具有更低的维度。

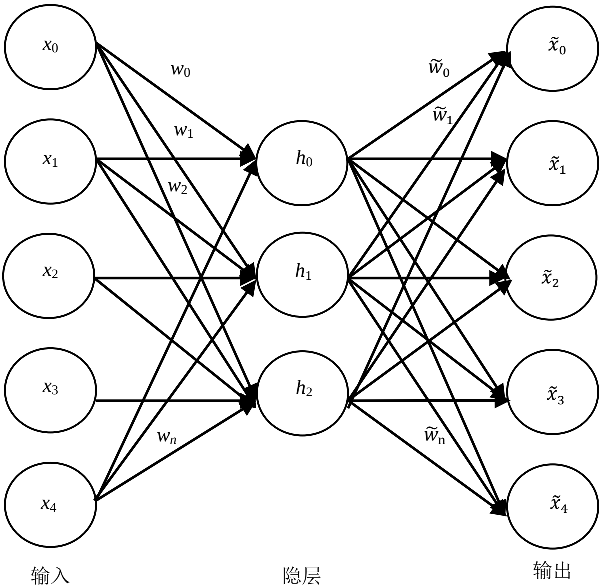


图 5.5　自编码器神经网路示意图

用隐层数据替代输入数据，既可以尽可能地保证信息量的完整，又可达到数据的降维。编码器的意义就在于充分使用隐层的输出，在信息量无损或者损失较小的前提下，获取数据低纬度表示。

自编码器使用无监督的方式实现模型的构建，其建模流程如图 5.6 所示。

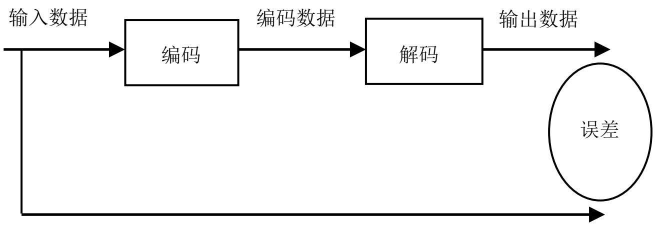


图 5.6　自编码器训练示意图

自编码器期望解码后的数据与输入数据尽可能相似，因此，以两者的误差平方和作为损失函数，运用反向传播算法求解使损失函数最小的权值 w_0~w_n。不难看出，自编码器的建模过程不需要标记训练数据，是一种无监督特征提取与降维的过程。

大多数读者对反向传播网络，即 BP（Backpropagation）神经网络耳熟能详。然而，它并不是像自编码器、限制玻尔兹曼机那样属于某种深度学习的方式，而是一种求解参数的算法。因为多数神经网络的参数是通过该算法求解，所以通常统一称其为 BP 神经网络。BP 算法极大提高了多层神经网络中参数的求解速度，使得训练多层神经网络成为可能。其思想和第 4 章介绍的梯度下降法基本一致，可以理解成梯度下降的思想在多层神经网络上的应用。

稀疏自编码器是指在编码的过程中加上稀疏性限制，以获得具有更多“0”值的编码数据。

如图 5.7 所示，当训练的过程中加入 L1 正则等稀疏性条件时，会使编码数据产生更多的“0”值。这种稀疏的性质使编码器抽取的特征具有更好的泛化效果。

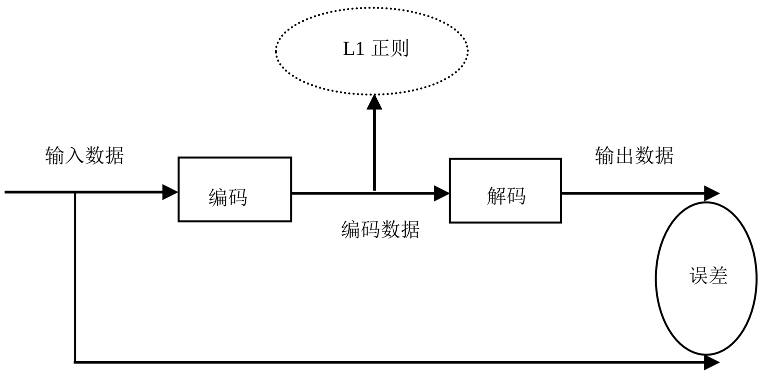


图 5.7　稀疏自编码器训练示意图

降噪自编码器，在输入数据中混入噪声数据，从而提升自编码器的抗噪声能力。如图 5.8 所示，噪声数据和训练数据一起编码与解码，同时，仅使用“非噪声数据”来衡量编码误差的优劣。显而易见，在该场景下，如果训练出的自编码器依然能够有效地编、解码，那么说明整个模型具有一定的抗噪能力。

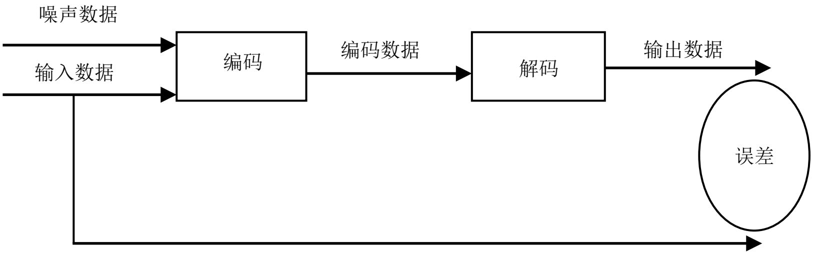
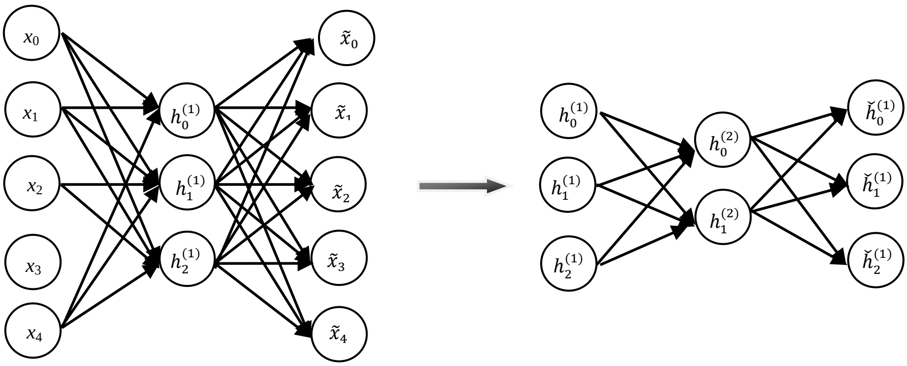


图 5.8　降噪自编码器示意图

栈式自编码器，如图 5.9 所示，当单隐层神经网络对训练数据进行一次编码之后，获取一阶特征的编码数据；再次编码可以得到数据的二阶特征；同理，多次编码之后即可得到高阶特征。以人脸识别为例，一阶特征会学习一些线条，二阶特征会学习脸部棱角，更高阶的特征则会学习像“鼻子”“嘴巴”这样的特征。栈式自编码器模型本身就是由一系列的自动编码器组合而成，并且这些自编码器都是各自独立训练的。

如图 5.9 所示，两个独立的自编码器，前一个自编码器的编码结果作为后一个编码器的输入数据。两个自编码器独自训练，最后合成栈式自编码器。



(a)　　　　　　　　　　　　(b)

图 5.9　栈式自编码器训练示意图

如图 5.10 所示，一个两层栈式自编码器，每层编码数据分别代表数据的一阶和二阶特征。同理，多层栈式自编码器由多个独立的单层自编码器组合而成，用以获取数据的更高阶特征。

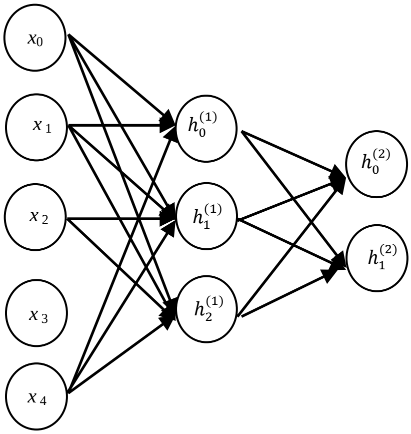


图 5.10　栈式自编码器结构示意图

以上依次介绍了单层自编码器、稀疏自编码器、降噪自编码器以及栈式自编码器。各种自编码器的根本作用都是为了提取数据的各阶特征，实际应用中常与数据挖掘中所述的各算法结合，以高效处理数据。

如图 5.11 所示，用自编码器产生的编码数据作为训练集来构建模型。在某些场景中，用编码数据替代原始数据往往能获得更好的效果。除此之外，栈式自编码器已经具备了神经网络的雏形，也可以将其看成后续的多层神经网络的基础。

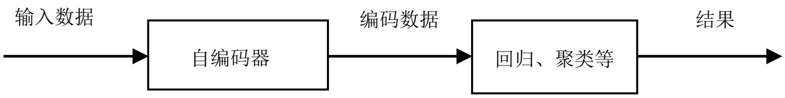


图 5.11　自编码器结合数据挖掘算法示意图

5.1.3　限制玻尔兹曼机

RBM（限制玻尔兹曼机）是一种基于数据集概率分布构建而成的单隐层神经网络模型。这种思想与第 4 章提到的模型聚类类似。RBM 是在 1986 年提出来的，但是由于求解困难，RBM 一直处在学术理论研究阶段，没有实际的工程应用。直到 Geoffrey Hinton 于 2000 年初发明了对比散度算法，极大降低了 RBM 模型建模的运算量以及复杂度，才使得 RBM 模型得到推广。此外，由于 Geoffrey Hinton

于 2006 年提出的 DNB（深信度网络）是以 RBM 为基石而构建的，这也带动了对 RBM 的研究。

限制玻尔兹曼机涉及诸多领域的理论知识，为了降低读者的学习梯度，更好地理解所述内容，本节按照玻尔兹曼机、限制玻尔兹曼机，ISING 模型、能量函数的顺序逐步讲解，并以建模思路为主、推导公式为辅的方式讲解相关各领域的知识。

与自编码器类似，BM（玻尔兹曼机）也是提取特征的一种方式，它是包含一个可见层和隐层、具有对称链接结构的神经网络，并且神经元的输出只有 1 和 0 两种状态，分别代表"激活"和"未激活"。BM 隐层神经单元的数量小于可见层，隐层的神经元可以看成可见层神经元的压缩表示，这种"压缩表示"的方式与自编码器类似，不过两者背后所依据的原理截然不同。BM 神经网络结构如图 5.12 所示。

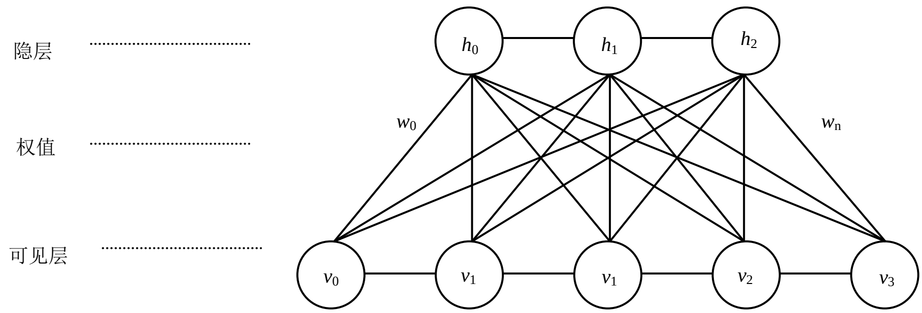


图 5.12　玻尔兹曼机示意图

如图 5.12 所示，BM 具有很强的无监督学习能力，但它的建模与计算过程却异常困难。于是在 BM 的基础上学术界又提出了 RBM（限制玻尔兹曼机）的概念，如图 5.13 所示。

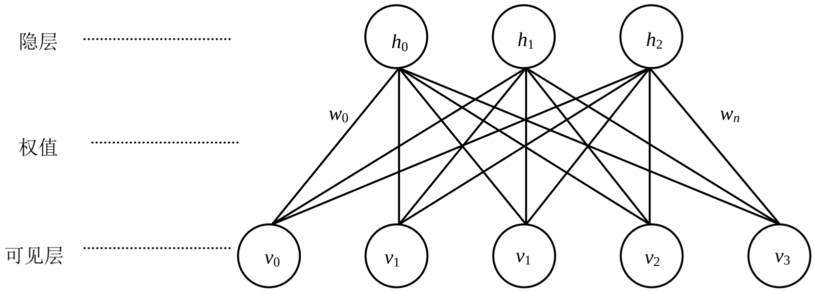


图 5.13　限制玻尔兹曼机示意图

与 BM 相比，RBM 去掉了隐层与可见层的层内的连接，其他的性质与 BM 相同。这种改动极大地方便了整个建模过程。

至此，读者应对 RBM 已有直观浅显的了解。RBM 可用于降维、分类等多种场景。比如，如果把 RBM 的隐层看做可见层的压缩表示，则 RBM 具有和编码器类似的降维效果；如果把 RBM 的隐层看做数据集各样本的类别，则 RBM 具有和 Softmax 类似的分类效果。接下来我们将更深一步地讲解 RBM 的原理和训练过程，以此说明在上述场景中的 RBM 的使用是合理的。

首先，如果隐层是可见层的另一种高阶特征或者类别输出，那么 RBM 的可见层与隐层应该具有一定的稳定关系。如果我们换个角度来看 RBM，把它看成由气体分子组成的热动力能量系统，那么可见层以及隐层中的神经元就是气体分子，那么，能量系统的最终状态必然属于或者接近稳定状态。因此，我们可以把 RBM 近似地看成一种处于平衡状态的能量系统。

在一定温度下，当热动力系统处于或接近平衡状态时，该系统的能量的概率分布属于玻尔兹曼分布。如果能用神经元的值和权重来表示系统的能量，就可以代入玻尔兹曼分布，获得系统能量的概率分布公式。之后，基于极大似然估计的思想（已发生的事物概率最大），求解该公式便可得到 RBM 系统中的参数。

如上所述，RBM 这种基于能量的模型，其可见变量 v、隐藏变量 h 所组成的系统的能量可以定义如下。

$$E(v,h,w) = -\sum_{i,j} w_{i,j}\, v_i h_j - \sum_i b_i v_i - \sum_j a_j h_j \tag{5.4}$$

其中 v、h 分别为可见层与隐藏层神经元的值，即 0 或 1；w 为权重，b 和 a 分别为可见层与隐藏层的偏置。将能量函数代入玻尔兹曼分布，可得到如下公式。

$$P(v,h) = \frac{e^{-E(v,h,w)}}{Z(w)} \tag{5.5}$$

其中 $Z(w)$是配分函数。接下来再通过最大化似然函数，并利用对比散度的算法求解参数，即可求得未知参数。为什么可以如此定公式 5.4 的能量模型义？我们接下来向读者介绍 ISING 模型，有助于帮助理解能量函数蕴含的现实意义。

ISING 模型最早由 Wilhelm Lenz 在 1920 年提出，该理论表述简单、内涵丰富且应用广泛。ISING 是为了研究磁铁在升到一定温度时磁性消失，而降到临界温度以下又会恢复磁性的现象而诞生的。假设磁铁由规则排列的小磁针构成，磁针的上下两个方向代表磁极，相邻的小磁针会在外界温度的作用下无序且剧烈转动，相互干扰导致磁性增强或者抵消。临界状态即指在温度的影响下，磁铁磁针的转动恰好抵消了所有磁性（也就是磁性消失时）的状态。该场景如图 5.14 所示。

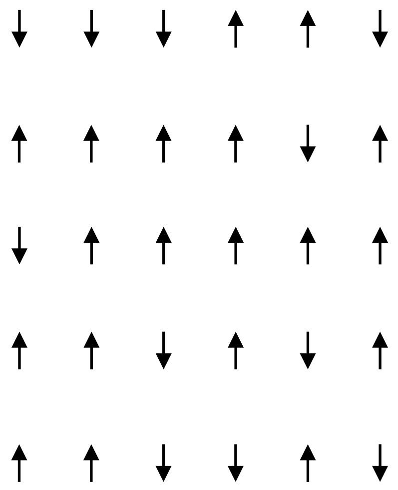


图 5.14　磁针的临界状态示意图

如图 5.14 所示，组成磁铁的小磁针具有两种状态，将磁针向上定义为+1，磁

针向下定义为−1。相邻磁针的相互作用决定了系统总能量的大小，当所有的磁针都是+1 或者−1 时，该系统的冲突最小，因而能量最小。我们可以这样来定义系统的总能量：如果相邻磁针的状态一致，则系统总能量−1；否则，总能量+1。能量公式定义如下。

$$E = -J\sum_{i,j} s_i\, s_j \; - H\sum_i^N s_i \tag{5.6}$$

其中 J 为一个能量耦合函数;H 表示外界磁场的强度,外界磁场向上 H 为正，否则为负。H 把外部影响加入模型中，当磁针与外界磁场影响一致时，冲动越小，总能量越低。

RBM 的神经元与 ISING 模型类似，都是只具有 0 或 1 两种状态属性。前者能量函数的定义正是由后者演化而来。

通过对自编码器以及限制玻尔兹曼机的讲解，读者不难发现，深度学习模型的建模过程不但包含了严谨的公式推导，而且数学公式的背后，还掺杂着许多对事物的哲学思考。罗列公式并非难事，而思想却无法仅用文字详述精髓。读者在学习这些数学公式的时候，可以去思考它们背后的哲理，加深对深度学习的认知和理解。

5.1.4　深度信念神经网络

深度信念神经网络，简称 DBN（Deep Belief Network），由 Geoffrey Hinton 在 2006 年提出。和栈式自编码器类似，DBN 是由若干层 RBM 和一层 BP 网络堆叠而成的深度神经网络[18]，网络结构如图 5.15 所示。

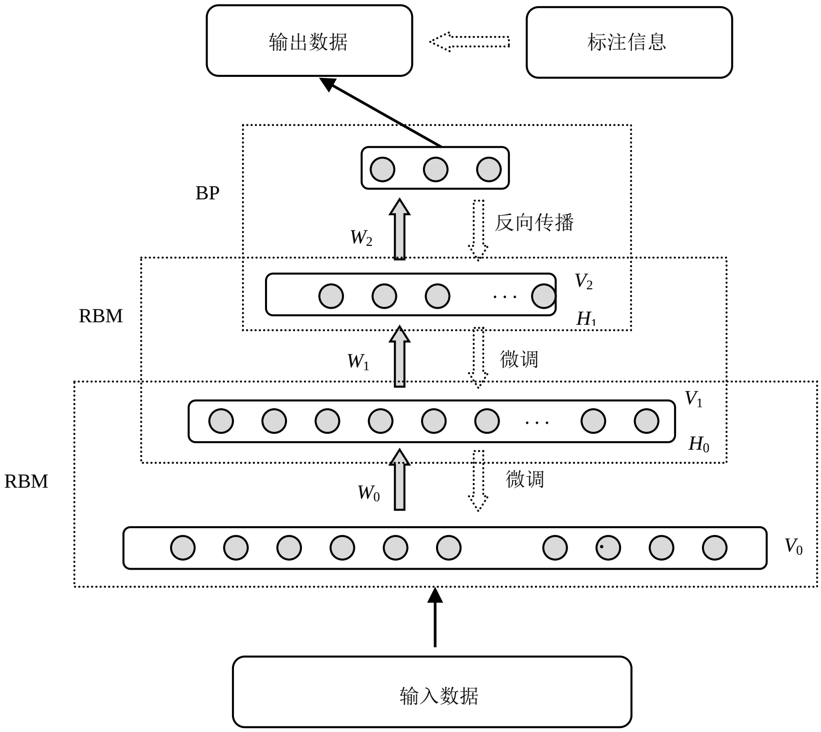


图 5.15　DBN 网络结构示意图

如图 5.15 所示，该 DBN 由两个 RBM 与一个 BP 网络构建而成。输入数据是已标注的数据集，用于后续的模型参数微调。DBN 的训练可以划分为以下两个步骤。

① 预训练阶段：以无监督的方式，单独训练各 RBM。其中底层 RBM 的隐藏层作为上层 RBM 的可见层，逐层训练。

② 微调阶段：最后一层 BP 神经网络接收 RBM 的隐藏层作为训练数据集。在经过步骤 1 之后，把整个 DBN 看成一个 BP 神经网络，RBM 训练的权值作为初始参数，运用 BP 算法再次迭代微调各层的权值参数。

DBN 的训练过程与栈式自编码器的训练十分相似。不同的是前者首先利用 RBM 训练初始参数，再用 BP 算法微调整个网络。这样做的好处是防止随机初始化参数导致的局部最优以及训练时间过长等问题。

自编码器与限制玻尔兹曼机具有相似的网络结构，但两种模型的建模思想截然不同。栈式自编码器和深度置信网络，分别使用自编码器和限制玻尔兹曼机堆叠而成，是同样具有相似网络结构、却包含不同建模理念的深度学习模型。所以，网络结构的形状并不是深度学习模型的本质特征，即使是相同的网络结构，也极有可能是不同的模型。只有深入了解建模过程所包含的思想，才能合理调整模型，更好地发挥其效用。

5.1.5　卷积神经网络

卷积神经网络，简称 CNN（Convolutional Neural Networks），由 Yann LeCun 于 1988 年在贝尔实验室工作期间发明，并随着不断的改良与演化，最终成为图像识别领域最重要的算法之一。本节在讨论 CNN 之前，会先抛出常规神经网络在处理图像时所存在的问题，并阐述 CNN 神经网络的特点；然后再详述相关概念及其训练过程；最后，列举并介绍 LeNet、AlexNet 的 CNN 网络结果模型。

（1）存在的问题

常规神经网络在处理图像时会造成过多的权值。以 $32 \times 32 \times 3$（32×32 像素，3 通道）的图像为例，输入数据具有 $32 \times 32 \times 3 = 3072$ 维，假设第一个隐层有 N 个神经元，则权值的个数为 $3072 \times N$。如果是一个更大尺寸的图像，比如 $1024 \times 1024 \times 3$，需要的权值的个数就相当多了。因此，用常规神经网络这种全连接的方式在处理图片时，不仅会因为权值过多而产生过拟合问题，在计算量上也会面临巨大的挑战。

图像数据由长、宽、深组成，其中深度指的是图像的通道。因此，图像可以看成是一个具有三维特征的数据体。然而，常规神经网络是把数据看成一维向量来组织构建的，这在图像处理中并不适用。图 5.16 引自 Stanford CS231n 课程，可以说明上述问题。

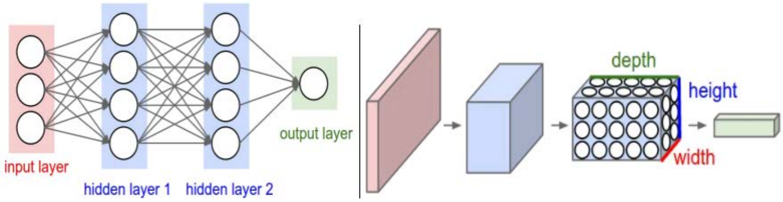


图 5.16　常规神经网络与 CNN 的对比

如图 5.16 所示，左侧为常规的神经网络结构，输入层、隐层的神经元都是一维排列。右侧是 CNN 神经网络的组织结构，输入和输出都是三维的数据体。读者可以对 CNN 与常规神经网络结构的异同先有个大概认识，并意识到后者在处理图像数据时存在的诸多问题，我们后面将对此做更加详细的阐述。

卷积神经网络解决了常规的神经网络在图像识别时所遇到的问题，提出了“局部感知野”“参数共享”“卷积操作”等一系列概念，在介绍卷积神经网络之前，我们先解释这些重要的概念。

（2）局部感知野

人类对图像的认知是一个由局部到全局、由部分到整体的过程。狭小范围内的像素是认知的基本单元，相互间的关联更大。比如，在欣赏一幅山水画时，脑海中出现的必然是画中的山、鸟、树木、河流等局部图像，最后再合成一幅全景。CNN 的提出正是从这里得到的启发：没必要从整体开始建模神经元，只需要局部感知，最后综合局部信息，实现全局感知。局部感知的区域就是“局部感知野”，可用图 5.17 表示。

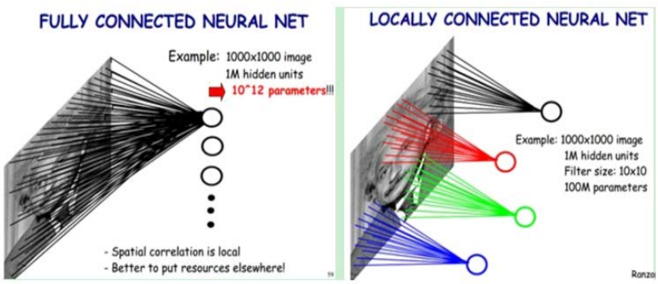


图 5.17　局部感知野示意图

如图 5.17 所示，左侧是一个全连接的神经网络，右侧则是依据“局部感知野”理念构建的局部连接的神经网络，显而易见，相比于全连接的方式，局部连接的好处是极大减少了权值的数量。

（3）参数共享

即使是基于“局部感知野”构建的局部连接网络，它所需要的权值的数量依然很大。如图 5.16 所示，假设隐层中的每个神经元对应一个局部碎片，并且有 100 个参数；如果把图像划分成 10000 份局部碎片，即有 10000 个神经元，那么就会有一百万个参数。那么我们如何再次减少参数的个数呢？“参数共享”的概念正是为了解决这样的问题而产生的。它的基本思想是如果把权值看成提取特征的一种方式，那么该方式与图像所在位置无关，因此，我们对所有局部图像碎片的特征提取都可以使用相同的参数。使用“参数共享”的方法可以把一百万个参数缩减到 100 个，极大提高了建模的可行性。

（4）卷积操作

卷积是 CNN 建模过程中最耗时且最核心的部分。所谓“卷积”，就是遍历“局部感受野”，并使用“共享参数”与局部数据做点积，生成新的网络隐层的过程。该过程如图 5.18 所示。

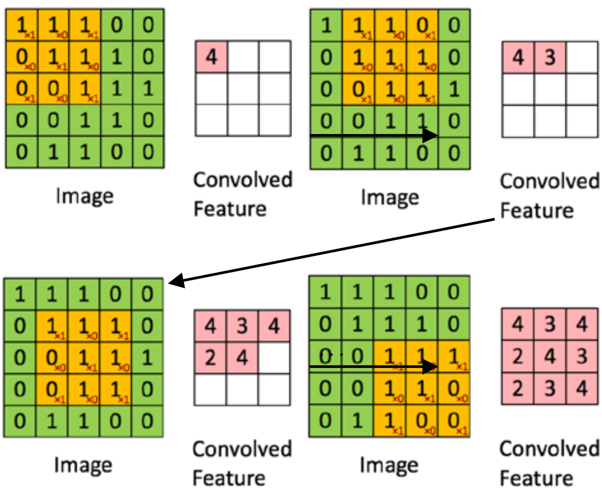


图 5.18　卷积过程示意

图 5.18 展示了一个 3×3 的卷积核在 5×5 的图像上做卷积的过程。感知野的尺寸是 3×3，步长是 1，输出的卷积特征是 3×3。在做卷积的过程中需要指定数据尺寸（W）、感知野的尺寸（F），步长（S），以及当感知野无法恰好穿过数据体时，做“零填充”的数量。已知上述参数后，输出数据体的尺寸可以通过$(W - F + 2P)/S + 1$ 计算。下面我们以图 5.19 为例说明零填充的作用。

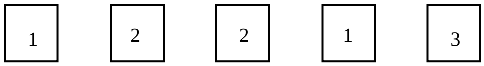


图 5.19　原始数据图

假设有一维空间数据如图 5.19 所示，输入数据的尺寸是 $W = 5$，感知野的尺寸是 $F = 3$，步长为 S=3，那么在第二次卷积操作时，恰好少了一位数据，在该场景中，可以对输入数据实施零填充。

如图 5.20 所示，经过零填充可以使得当步长 S=3 时，卷积核恰好穿过数据集，简化了卷积操作的实现。

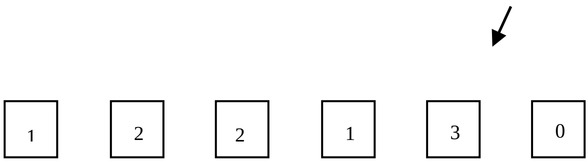


图 5.20　零填充示意图

（5）多卷积核

“局部感知野”的大小，或者说卷积核的大小对特征提取影响极大。直观而言，从多个角度或者粒度分割看待图像，往往能获得不同的信息。这就相当于不同的卷积核可以从不同的维度生成原图像的另一种表示。因此，建立多个大小不同的卷积核能更充分地提取图像特征，达到更优的建模效果。

如图 5.21 所示，两个卷积核分别生成了原图像的两种表示。通常在建模的过程中会设定上百个卷积核，并分布在不同的 GPU 上计算，在后面的 CNN 模型示例中，我们将了解多卷积核的应用。

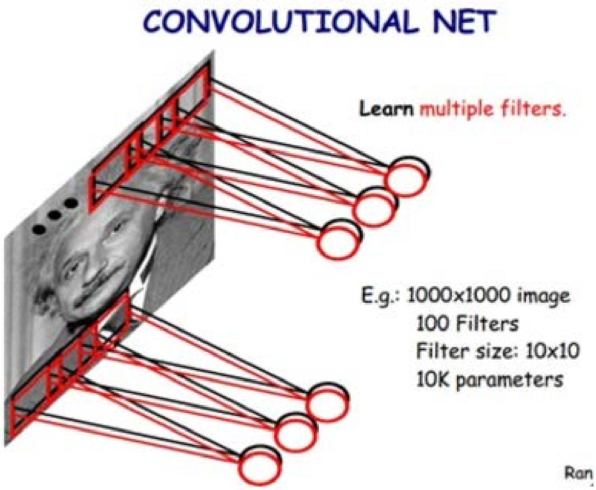


图 5.21　局部感知野示意图

（6）池化

在介绍池化的概念之前，先描述池化所要解决的问题。假设有一个 512 × 512 的图像，卷积核的大小是 8 × 8，步长为 1，对其做卷积操作。单卷积核操作时，可以得到(512 − 8 + 1) × (512 − 8 + 1) = 255 025 个神经元。如果设定 256 个卷积核，那么最终卷积的过程将得到 256 × 255 025 个神经元。这么多数量的神经元给建模带来的困难是很明显的，为了解决这一问题，我们引入“池化”的概念。

可以把“池化”操作视同在二维平面上，将图像划分为很多小区域，并使用该区域的统计特征来代表该区域的信息。比如最大值、均值等。

图 5.22 来自 Stanford CS231n 课程，它给定一个 2 × 2 尺寸的池化粒度，对 4 × 4 二维数据进行操作，选取区域内的最大值代表该区域，最终获得一个 2 × 2 的数据集。由此可知，“池化”操作减少了数据量，一定程度上合理地解决了上述问题。

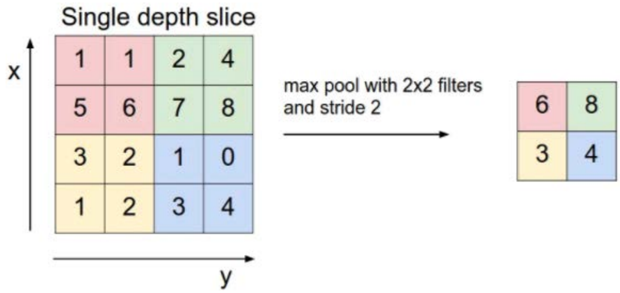


图 5.22　池化示意图

（7）卷积示例

上面我们分别介绍了 CNN 建模过程中“局部感知野”“参数共享”“卷积”“多卷积核”“池化”等重要概念，下面我们希望通过示例再对卷积做直观整体的介绍。

如图 5.23 所示，我们对长度为 H=5，宽度为 W=5，深度为 D =3 的图像做 CNN 建模。设定多卷积核个数为 2，即 w_0 与 w_1，大小均为 3×3。以步长 S=2 做卷积且进行“零填充”。第二步操作如图 5.24 所示。

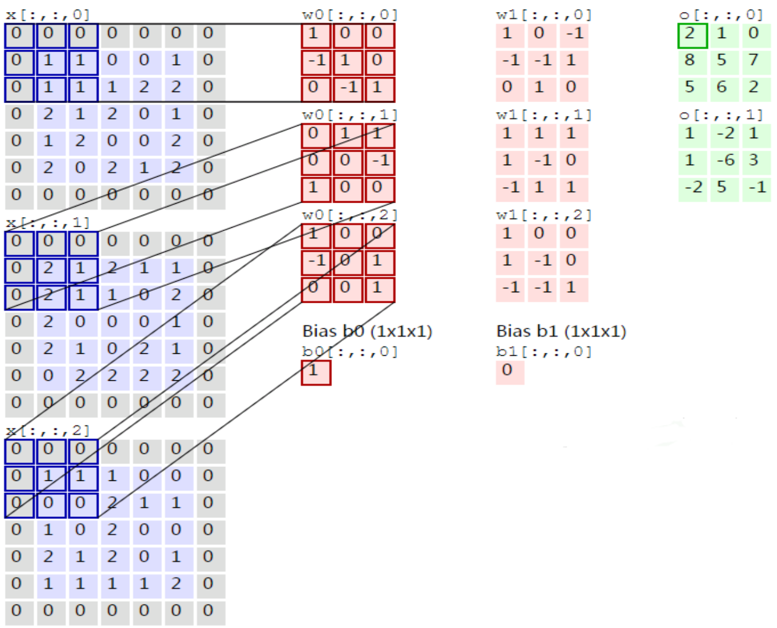


图 5.23　CNN 建模示意图

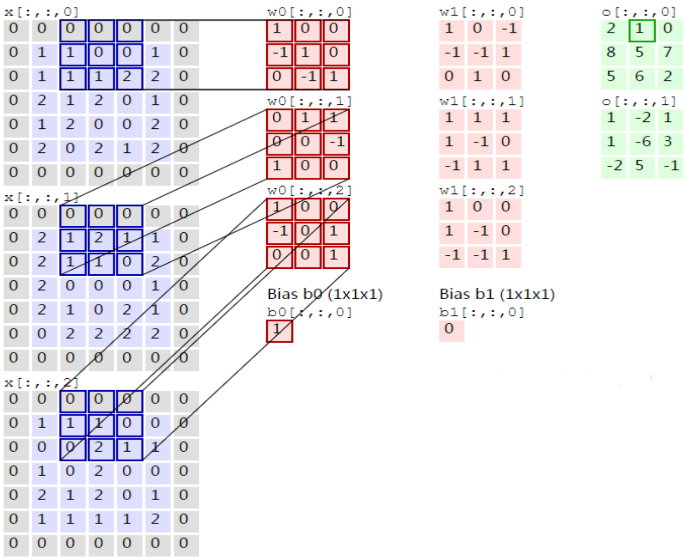


图 5.24 CNN 建模示意图

用步长 $S = 2$ 实现卷积操作，各数值结果如图 5.24 所示。当第一个卷积核操作完成后，依据第二个卷积核再次实施卷积操作，如图 5.25 所示。

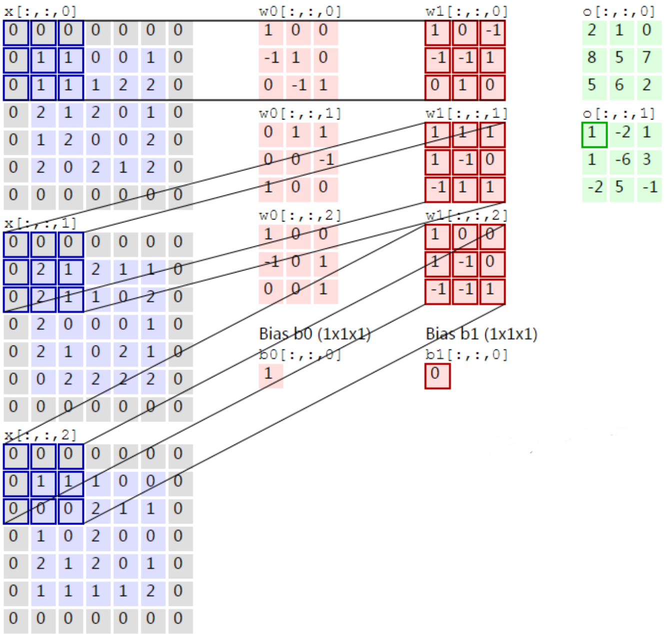


图 5.25 CNN 建模示意图

以上卷积最终输出 2 个 3×3 的数据集,两个卷积核代表从两个角度提取了原始图像的特征，这些代表原始图像特征的数据称为特征图。这些特征图将用于后续的池化、再次卷积、全链接等一系列过程。

（8）LeNet-5

20 世纪 90 年代，CNN 之父 Yann LeCun 创造了经典的 LeNet 模型，并成功地用于数字识别和邮政编码等领域。

如图 5.26 所示，LeNet-5 一共 6 层。输入数据是 32×32 的字母图片。C1 层是卷积层，一共有 6 个卷积核，要构造 28×28 的特征图。S2 层是下采样层，该层的特征图经过池化操作转换为 6 个 14×14 的特征图。C3 是卷积层，一共有 16 个卷积核，该层的特征图经过卷积操作构造出 10×10 的特征图。S4 层是下采样层，该层的特征图经过池化操作转换为 16 个 5×5 的特征图。C5 是卷积层，一共有 120 个卷积核，该层的特征图经过卷积构造出 1×1 的特征图。F6 层与 C5 层全连接,并输出到输出层。输出层的特征图最终通过 Softmax 回归来完成分类工作。

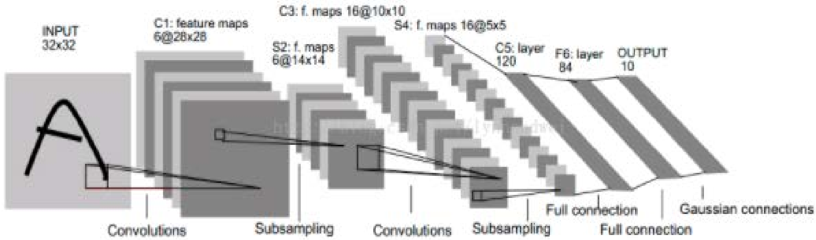


图 5.26 LeNet 网络结构示意图

（9）AlexNet

和 LeNet 相比，AlexNet 是后起之秀，它由 Alex Krizhevsky、Geoffrey Hinton 和 Ilya Sutskever 合力完成。该模型不负众望，在 2012 年 ImageNetILSVRC 竞赛中夺冠。

如图 5.27 所示，输入数据是 224×224 大小的 3 通道图像数据。第一层采用

96 个大小为 11 × 11 的卷积核实施卷积操作，并采用 2 × 2 的核做“最大池化”。第二层采用 256 个大小为 5 × 5 的卷积核实施卷积操作;依然采用 2 × 2 的核做“最大池化”。第三层与上一层全连接。第四层采用 384 个 3 × 3 的核实施卷积操作。第五层采用 256 个 3 × 3 的卷积核做卷积操作，采用 2 × 2 的核做“最大池化”。第五层之后就是全连接层以及 Softmax 层。

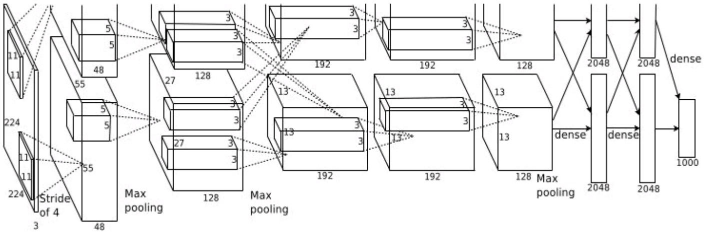


图 5.27　AlexNet 示意图

以上我们简要介绍了 LeNet、AlexNet 卷积神经网络模型，让读者更清晰地了解 CNN 模型的构建。除此之外，还有诸如 ZFNet、GoogleNet、VGGNet 等一系列经典的网络架构模型，每种模型都有独特的优势和相关的应用场景，感兴趣的读者可以自行研究这些经典的建模案例。

5.2　人工智能应用概况

5.2.1　**深度学习的历史**

20 世纪 80 年代，Gauss 和 Legendre 等杰出数学家的研究工作为神经网络理论的演化和最终诞生奠定了坚实的基础。

1943 年，心理学家 Warren McCulloch 和数理逻辑学家 Walter Pitts 在 *A logical calculus of the ideas immanent in nervous activity* 中提出了人工神经网络的概念，从此开始了人工智能研究的时代[17]。

1949 年，Donald Olding Hebb 出版 *The Organization of Behavior* 一书，并在书

中提出了“赫布理论”[19]。该理论诠释了学习过程中脑神经元所发生的变化，为后人效仿大脑运作模式、构建人工智能机器模型奠定了理论基础。

1950 年，Alan Mathison Turing 在 *Computing Machinery and Intelligence* 一文中提出著名的“图灵测试”[20]：即测试者与被测试者隔开，通过某些装置向被测试者提问，如果超过 30%的测试者不能确定被测者是人还是机器，那么这台机器就被认为具有“人类智能”。

1957 年，Frank Rosenblatt 就职于 Cornell 航空实验室时发明了“感知机”，并于 1958 年将感知机的相关成果发表在 *The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain* 一文中[16]。

1961 年，Frank Rosenblatt 出版 *Principles of Neurodynamics: Perceptrons and the theory of brain mechanisms* 一书，深入解释了感知机的背景假设和定理证明（比如算法的收敛性[21]）。

1969 年，Marvin Minsky 和 Seymour Papert 出版 *Perceptrons: an introduction to computational geometry* 一书[22]，分析了以感知机为代表的单层神经网络的功能和局限性，指出感知机无法解决“异或”等线性不可分的问题。该书产生巨大的影响，导致神经网络的研究进入寒冬时期。

1974 年，Paul Werbos 在哈佛大学的 Phd. 论文 *New Tools for Prediction and Analysis in the Behavioral Sciences* 中首次描述了使用BP算法训练神经网络的过程[23]。由于当时正值人工智能研究的低谷期，该论文并未受到重视。

1986 年，Geoffrey Everest Hinton、David E. Rumelhart 以及 Ronald J. Williams 三人合作发表了 *Learning representations by back-propagating errors* 一文，使得 BP 算法获得广泛认可[24]。与此同时，人工智能的研究也进入“文艺复兴”时代。

1989 年，Yann LeCun 发表 *Backpropagation Applied to Handwritten Zip Code Recognition* 一文，提出了卷积神经网络，以及识别邮编手写体字符的 LeNet 模型[25]。Yann LeCun 本人也因此被称为“卷积神经网络之父”。

1995 年，Corinna Cortes 和 Vladimir Vapnik 发表了 *Support-Vector Networks* 一文，详解 SVM 建模算法[26]。虽然 SVM 需要手动指定参数，但是它能有效处理线性不可分的数据集，且训练速度优于常规神经网络。同时，学术界也发表了诸如 Boosting 等一些把弱分类器转化为强分类器的文章。这些状况都直接导致研究方向的偏移，使得深度神经网络的发展再次陷入低谷。

2002 年，Geoffrey Everest Hinton 在 *Contrastive Divergence Training Products of Experts by Minimizing CD* 一文中详述了“对比散度”算法[27]。该算法加快了 RBM 模型的构建，使得 RBM 真正变得可用；并且也为后面 DBN 模型的诞生起到了至关重要的作用。

2006 年，Geoffrey Everest Hinton、Simon Osindero 以及 Yee-WhyeTeh 在 *A fast learning algorithm for deep belief nets* 一文中提出了“深度信念网络”[28]。该网络模型提出降低隐藏层训练的难度，极大提升了最终效果和训练速度。

2006 年至 2017 年，虽然没有更新的突破性的研究理论涌现，但是，学术界正不断与工业界融合，将已有的人工智能理论应用在各行各业，极大改变了人们的日常生活。无论是理论研究还是工程应用，抑或社会对人工智能的接受程度，都处在前所未有的热度中，我们将亲眼见证人工智能的发展。

如图 5.28 所示，1958 年以感知机模型为代表，深度学习首次兴起。1969 年，由于 Marvin Minsky 等人指出感知机在处理“非线性数据”时存在的缺陷，导致了 1969~1985 年神经网络研究的低谷期。1986~1990 年，BP 算法和 CNN 的提出再次点燃深度学习研究的热潮。1995 年，SVM 模型和 Boosting 方法的出现，导致众多学者研究方向的转移。2006 年，随着 DBN 模型的提出和工业界对人工智能的重视，深度学习的研究再次成为热点并蓬勃发展。

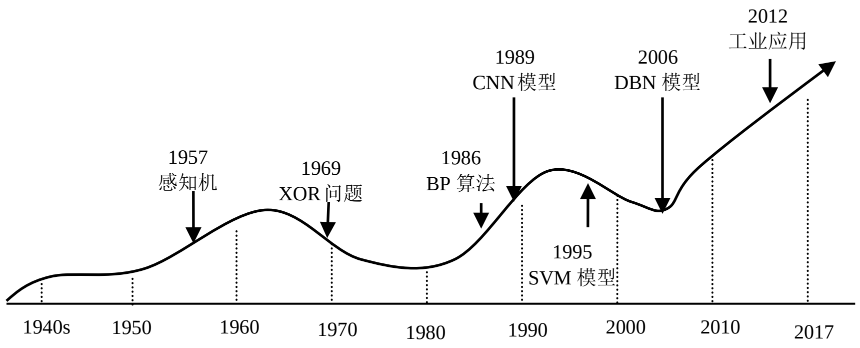


图 5.28　深度学习发展示意图

5.2.2　人工智能的杰作

（1）谷歌和 AlphaGo

AlphaGo 是 2014 年由谷歌旗下的 DeepMind 公司开发的一款人工智能围棋程序。它使用深度学习等技术，在完全自主学习的情况下实现对弈水平的逐步提升，并且在后续的博弈中，战绩卓著，影响巨大，成为“人工智能”的代名词。

我们回顾一下历史上人机对弈的大战。

19 世纪 90 年代末，随着计算机技术的发展，人机对弈程序已初露端倪。

1997 年，IBM 的“深蓝”电脑击败俄籍国际象棋冠军加里·卡斯帕罗夫。

2012 年，Zen 围棋程序在让 5 子和让 4 子的情况下，两次击败日本九段棋手武宫正树。

2014 年，CrazyStone 在让 4 子的情况下，击败日本九段棋手石田芳夫。

2014 年，单机版 AlphaGo 在和 CrazyStone 以及 Zen 等程序的 500 局对战中仅输一局。

2015 年 10 月，AlphaGo 以 5:0 击败欧洲围棋冠军樊麾。

2016 年 3 月，AlphaGo 以 4:1 的成绩战胜世界围棋冠军、职业九段选手李世石。

2016 年 12 月，AlphaGo 以 Master 的网名，在各大围棋对战平台，横扫众多

世界级冠军。

AlphaGo 为何能在人机对弈和机器对弈中独占鳌头？这归功于包括蒙特卡洛树搜索、估值网络、策略网络等技术的融合运用。其中，后两者是一种多层 CNN 神经网络模型。得益于深度学习理论的发展，AlphaGo 获得了更强的智能。

（2）谷歌和自动驾驶

谷歌无人车（如图 5.29 所示）是谷歌街景共同发明人 Sebastian Thrun 领导的项目，旨在打造一款不需要驾驶者操作，就能启动、行驶以及停止的具备人工智能的车辆。

2009 年谷歌曝光了自动驾驶汽车的雏形。

2010 年 9 月，谷歌在官方博客宣布正在研发自动驾驶汽车。

2011 年 10 月，谷歌在内华达州和加州莫哈韦沙漠试验场测试汽车。

2012 年 4 月 1 日，谷歌向外界展示了具有自动驾驶技术的赛车。

2012 年 5 月，谷歌获得了美国首个自动驾驶车辆许可证。



图 5.29　2009 年谷歌曝光的无人车雏形

自动驾驶的实现涵盖传感器技术、定位技术、图像识别技术等多学科领域。可想而知，无人车要上路行驶，必须能清楚地识别直道或弯道、限速警示牌和周围车辆与环境等。图像识别在整个技术体系中占据着毋庸置疑的重要位置。而 DBN、CNN 等深度学习模型恰是图像识别的利器，它们的发展与完善加速了无人驾驶时代的到来。

（3）谷歌和 Google Home

Google Home（如图 5.30 所示）是一款在家庭场景中使用的，携带语音助手功能的硬件设备。用户可以使用“请把厨房的灯关闭”“播放音乐”等自然语言的交互方式，远程操控音箱、灯光等设备；或者用“今天天气如何”“今天的热点新闻”等方式咨询各种问题。



图 5.30　Google Home 设备

2016 年 5 月 19 日，谷歌在 I/O 开发者大会上，首次发布了内置扬声器的语音激活设备 Google Home；并于 2016 年月 10 月 5 日正式推出智能音箱 Google Home。

（4）微软 Cortana

Cortana（如图 5.31 所示）是微软在 Windows Phone 8.1、Microsoft Band、Windows 10、iOS、Android 上推出的一款智语音助手。与 Google Home 一样，可以使用自然语音询问天气、交通、赛事等信息。

2014 年 4 月在美国旧金山微软 Build 大会上，微软副总裁 Joe Belfiore 首次展示了 Cortana。

2014 年末至 2015 年初向全球英语版 Windows Phone 8.1 的用户开放。

2015 年 12 月 9 日，微软正式发布 iOS 和 Android 版本的 Cortana。

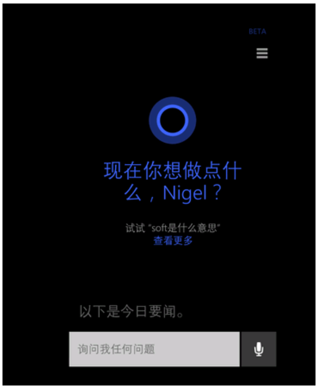


图 5.31　微软 Cortana 交互界面

（5）苹果 Siri

Siri（如图 5.32 所示）是大多数“果粉”非常熟悉的一款智能语音助手。苹果公司在 iPhone 4S、iPad 3 及以上版本的手机和 Mac 上都嵌入了该软件。Siri 可以让用户更方便快捷地操作苹果设备，比如，只要对着 iPhone 说“帮我定明天早上 8 点的闹钟”即可方便地实现定时功能。



图 5.32　苹果 Siri 交互界面

2007 年，Siri 公司成立，创始人有 Dag Kittlaus、Adam Cheyer 以及 Tom Gruber。

2008 年 10 月 13 日，Siri 公司第一轮融资达到 850 万美元。

2010 年 4 月 28 日，苹果完成对 Siri 的收购。

2011 年 10 月 29 日，Siri 成功移植到 iPhone 4 上。

2016 年 6 月 13 日，苹果在 WWDC 开发者大会上发布了 Siri 产品的新功能。

（6）亚马逊 Echo

Echo（如图 5.33 所示）与 Google Home 属于同一类竞品，在物联网智能家居领域中的地位无人可及，是首屈一指的智能产品。不论是 Google Home，还是百度基于 DuerOS 打造的多款智能硬件，都受到了 Echo 的巨大影响。Alexa 是 Echo 内置的智能语音助手软件，也是该音箱的唤醒词。如果你想听音乐，只要对着 Echo 说“Alexa，放一首音乐听听吧”即可实现乐曲播放功能。



图 5.33　Echo 产品实体图

2014 年 11 月 6 日，Echo 在亚马逊官网上线。2015 年，该产品占据音箱市场高达 25%的份额。2016 年，销量超过 650 万台。Echo 的火爆，从一个侧面展现出用户对智能硬件产品的接受程度，也反映了物联网和人工智能对生活的变革。

（7）百度 DuerOS

DuerOS 是由百度开发的一款与 Alexa 相似的智能语音助手。基于 DuerOS 系统，开发者可以在各种场景中打造多样的智能产品。比如在“最强大脑”中具有非凡能力的机器人“小度”，以及“手机百度”App 软件中嵌入的语音助手“度秘”等。同时，DuerOS 在与“小鱼在家”的深度合作中，打造了具有聊天对话、信息查询等多种能力的机器人（如图 5.34 所示）。



图 5.34　搭载 DuerOS 的小鱼在家机器人

百度还发布了嵌入 DuerOS 的智慧芯片。该芯片嵌入了麦克风，可以直接搭载到硬件产品中，让其即刻成为可以对话的智能硬件；同时，也允许合作的硬件厂商基于该芯片开发自己的特色产品。这种方式极大降低了硬件智能化的成本，从而让更多产品具有“智能”。

2015 年 9 月 8 日，百度创始人李彦宏在百度世界大会上演示了语音虚拟助手“度秘”。

2016 年 4 月，“度秘”入驻 KFC，帮助用户点餐。

2016 年 8 月，与篮球解说员杨毅老师解说澳大利亚与立陶宛的奥运 1/4 决赛。

2017 年 1 月 5 日，百度宣布了 DuerOS 系统，旨在打造人工智能的一体化平台。

2017 年 2 月 16 日，百度 COO 陆奇宣布成立“度秘事业部”，围绕 DuerOS 打造智能生态。

不难看出，2016 年前后，是人工智能技术走向应用，改变生活的极其重要的时间点。从 Alexa、DuerOS 以及 Siri 等智能语音平台，到 Echo、Siri、小鱼在家等智能产品，国内外的巨头公司逐步意识到人工智能所能带来的巨大变革，纷纷投入到该领域的探索与耕耘中。人工智能以语音助手的方式落地，就像刚出生的婴儿，伴随着技术的演化和需求的推动，它也必然将具有识别图形、识别音色、识别动作甚至识别语气等更加强大的智能。

5.3 实例讲解

5.3.1 学习识别手写数字

（1）背景说明

MNIST 是从 NIST（National Institute of Standards and Technology）的“Special Database 3”和“Special Database 1”中抽取的一个数据库子集（如图 5.35 所示）。该数据库共有 70,000 份带有标签的、28×28 尺寸的黑白手写数字图像，其中 60,000 份用于训练，10,000 份用于测试[29]。

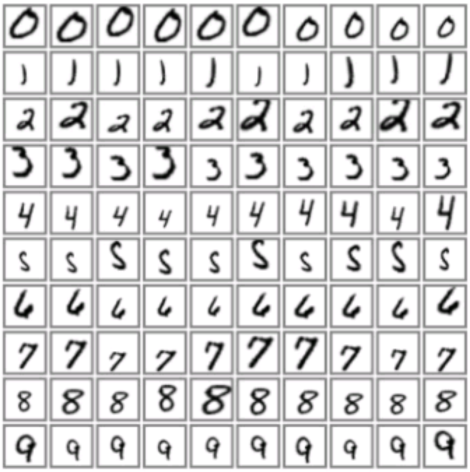


图 5.35　MNIST 数字图像

MNIST 是各种图像建模中的常用数据库。许多机构尝试用 SVM、KNN、CNN 等多种方法对其建模，以提高图像识别的准确率。Yann LeCun 的网站上有相关研究的信息汇总和论文链接地址，有兴趣的读者可以深入研究。

TensorFlow 是谷歌开源的一款机器学习的开源软件库。该库封装了 Softmax、CNN、DBN 等各种建模函数，简单易用，且支持 Python、C++等多种开发语音，是当前最受欢迎的机器学习库之一[30]。

本实例中采用 TensorFlow 实现 Lenet5 模型，训练 MNIST 数据集。各个参数的取值参考 Caffe 中的 Lenet5 范例。读者需要注意 Caffe 中 Lenet5 的参数与 Yann LeCun 教授原始论文中的模型有所差异（参见 5.1.5 节）。

（2）建模流程

① 使用 input_data.py 脚本读取 MNIST 数据。该脚本由 TensorFlow 官网提供，封装了下载以及读取 MNIST 数据库中的数据的代码。

② 构建 Lenet 模型如图 5.36 所示。

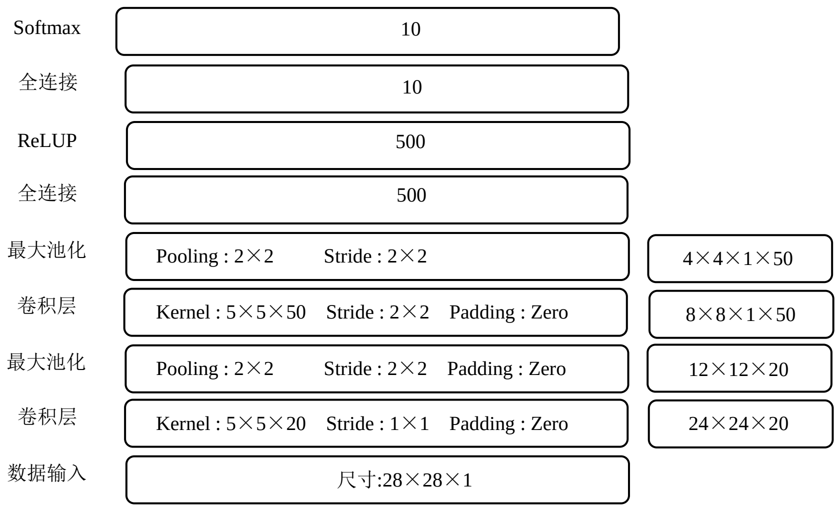


图 5.36　MNIST 手写字体识别网络结构图

- 输入大小为 28 × 28，深度为 1 的图片执行卷积操作。卷积核大小：5 × 5；卷积核数量：20；横向和纵向步长 1 × 1；
- 采用零值填充。经过该卷积操作，生成大小为(28−5+1) × (28−5+1)，深度为 20 的图片；
- 执行最大池化。池化窗 2 × 2；横纵向步长 2 × 2。生成大小为(24 / 2) × (24 / 2)，深度为 20 的图片；
- 执行卷积操作。卷积核大小：5 × 5；卷积核数量：50；横向和纵向步长 1 × 1；

- 采用零值填充。经过该卷积操作，生成大小为$(12-5+1)\times(12-5+1)$，深度为 50 的图片；
- 执行最大池化。池化窗 2×2；横纵向步长 2×2。生成大小为$(8/2)\times(8/2)$，深度为 50 的图片。
- 执行全连接操作。神经元数量：500。执行 ReBLU 操作。
- 执行全连接操作，神经元数量：10。执行 Softmax 回归。

③ 准确率验证。采用 input_data.py 脚本获取 MNIST 的测试数据集，进行准确率验证。

（3）建模代码

程序代码由 input_data.py、train.py、lenet5.py 三个文件构成。其中 input_data.py 从 TensorFlow 官网的“MNIST 识别实例代码”下载；train.py 包含了程序的启动入口函数；提取了 Lenet5 的建模代码，单独放在 lenet5.py 中。组织结构如图 5.37 所示。

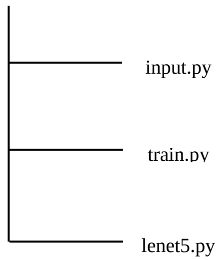


图 5.37　识别手写字体代码目录结构

train.py

```
#!usr/bin/python
# -×- coding:utf-8 -×-
import input_data
import tensorflow as tf
import lenet5

# mnist 数据集
mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)
```

```
    # 图片转为1D向量的长度
    image1D = 28×28

    # 手写数字图片类别(1、2、3、4...10)
    classes = 10
    batchSize = 100;

    # 分批训练数据，设定占位符，代表某一批次
    x = tf.placeholder(tf.types.float32, [None, image1D])
    y = tf.placeholder(tf.types.float32, [None, classes])

    # 构建模型
    pred = lenet5.lenet5(x)

    # 损失函数及最优化求解
    cost   =   tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits
(pred, y))
    train  =tf.train.AdamOptimizer(1e-4).minimize(cost)

    # 准确读评估
    correct_pred = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
    #初始化所有的变量
    init = tf.initialize_all_variables()

    # 开启一个训练
    withtf.Session() assess:
        sess.run(init)
        step = 1
        whilestep×batchSize<200000:
        batchXs, batchYs = mnist.train.next_batch(batchSize)
        # 获取批数据
        sess.run(train, feed_dict={x: batchXs, y: batchYs})
        ifstep % 100 == 0:
        # 计算精度
        acc = sess.run(accuracy, feed_dict={x: batchXs, y: batchYs})
        # 计算损失值
        loss = sess.run(cost, feed_dict={x: batchXs, y: batchYs})
        print  "Step  =  "  +  str(step)  +  ",  BatchLoss  =  "  +
"{:.6f}".format(loss) + ", TrainAccuracy= " + "{:.5f}".format(acc)
        step += 1
    print "OptimizationFinished!"

    # 计算测试精度
    print "TestAccuracy:", sess.run(accuracy, feed_dict={x: mnist.test.
```

```
images[:256], y: mnist.test.labels[:256]})

    lenet5.py
    #!usr/bin/python
    # -×- coding:utf-8 -×-
    import input_data
    import tensorflow as tf
```

#lenet5

```
    deflenet5(x):
    # 第一层 :Data (-1 代表批量, 28、28 代表图片大小，1 代表图片深度)
    x = tf.reshape(x, shape = [-1, 28, 28, 1])

    # 第二层 : 卷积
    conv1Weights  =  tf.Variable(tf.random_normal([5,  5,  1,  20],
stddev=1e-4))
    conv1Biases  =tf.Variable(tf.random_normal([20], stddev=1e-4))
    conv1 = tf.nn.bias_add(tf.nn.conv2d(x, conv1Weights, strides=[1, 1,
1, 1], padding='SAME'), conv1Biases)

    # 第三层 : 池化
    pool1 = tf.nn.max_pool(conv1, ksize=[1, 2, 2, 1], strides=[1, 2, 2,
1], padding='SAME')

    # 第四层 : 卷积
    conv2Weights  =  tf.Variable(tf.random_normal([5,  5,  20,  50],
stddev=1e-4))
    conv2Biases  =tf.Variable(tf.random_normal([50], stddev=1e-4))
    conv2   =   tf.nn.bias_add(tf.nn.conv2d(pool1,   conv2Weights,
strides=[1, 1, 1, 1], padding='SAME'), conv2Biases)

    # 第五层 : 池化
    pool2 = tf.nn.max_pool(conv2, ksize=[1, 2, 2, 1], strides=[1, 2, 2,
1], padding='SAME')

    # 第六层 : 全连接
    poolShape = pool2.get_shape().as_list()
    shapeNum  =poolShape[1]×poolShape[2]×poolShape[3]
    reshaped = tf.reshape(pool2, [-1, shapeNum])
    ip1Weights   =   tf.Variable(tf.random_normal([shapeNum,   500],
stddev=1e-4))
    ip1Biases  =tf.Variable(tf.random_normal([500], stddev=1e-4))
    ip1 = tf.matmul(reshaped, ip1Weights) + ip1Biases
```

```
relu1 = tf.nn.relu(ip1)

# 第七层 : 全连接
ip2Weights = tf.Variable(tf.random_normal([500, 10], stddev=1e-4))
ip2Biases  =tf.Variable(tf.random_normal([10], stddev=1e-4))
ip2 = tf.matmul(relu1, ip2Weights) + ip2Biases
returnip2
```

运行结果如图 5.38 所示，经过 2000 次的训练，该模型对“手写字体”的识别准确率达到 90%。在测试集上准确率达到 94.15%。

```
[root@localhost lenet5]# python train.py
Extracting /tmp/data/train-images-idx3-ubyte.gz
Extracting /tmp/data/train-labels-idx1-ubyte.gz
Extracting /tmp/data/t10k-images-idx3-ubyte.gz
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz
I tensorflow/core/common_runtime/local_device.cc:25] Local device intra op parallelism threads: 1
I tensorflow/core/common_runtime/local_session.cc:45] Local session inter op parallelism threads: 1
Step = 100, Batch Loss = 2.272235, Train Accuracy= 0.17000
Step = 200, Batch Loss = 1.518637, Train Accuracy= 0.53000
Step = 300, Batch Loss = 0.763219, Train Accuracy= 0.78000
Step = 400, Batch Loss = 0.664969, Train Accuracy= 0.78000
Step = 500, Batch Loss = 0.850403, Train Accuracy= 0.73000
Step = 600, Batch Loss = 0.713781, Train Accuracy= 0.83000
Step = 700, Batch Loss = 0.377293, Train Accuracy= 0.83000
Step = 800, Batch Loss = 0.512659, Train Accuracy= 0.86000
Step = 900, Batch Loss = 0.313179, Train Accuracy= 0.88000
Step = 1000, Batch Loss = 0.369643, Train Accuracy= 0.87000
Step = 1100, Batch Loss = 0.424126, Train Accuracy= 0.86000
Step = 1200, Batch Loss = 0.415794, Train Accuracy= 0.90000
Step = 1300, Batch Loss = 0.340066, Train Accuracy= 0.90000
Step = 1400, Batch Loss = 0.365505, Train Accuracy= 0.90000
Step = 1500, Batch Loss = 0.252521, Train Accuracy= 0.90000
Step = 1600, Batch Loss = 0.198899, Train Accuracy= 0.93000
Step = 1700, Batch Loss = 0.248384, Train Accuracy= 0.90000
Step = 1800, Batch Loss = 0.168542, Train Accuracy= 0.96000
Step = 1900, Batch Loss = 0.272401, Train Accuracy= 0.90000
Optimization Finished!
Test Accuracy: 0.941406
```

图 5.38　手写数字识别运行结果

5.3.2　让机器认识一只猫

（1）背景说明

CIFAR-10 和 CIFAR-100 是 Alex Krizhevsky、Vinod Nair 以及 Geoffrey Hinton 搜集整理的图片数据库。其中 CIFAR-10 包含了 60,000 幅 32 × 32 的彩色图像数据，50,000 用于训练，10,000 幅用于测试[31]，共有如图 5.39 所示的 10 个类别。

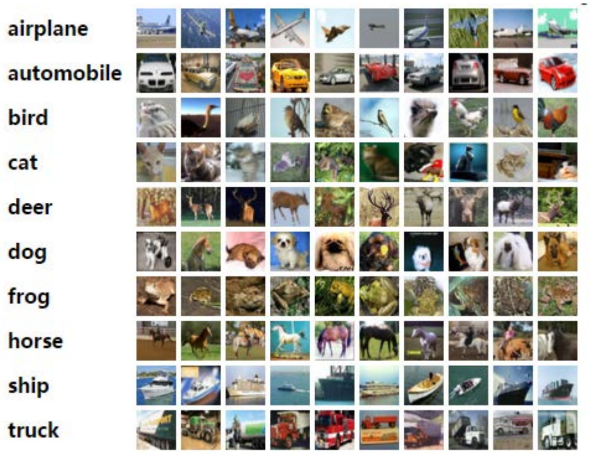


图 5.39　CIFAR-10 图像类别

CIFAR-10 数据库中的图像都是车、马、轮船等物体在现实场景中的图像。同时,分类都是非重叠的,不存在某一类型的图像同时属于两个及以上类别的情况。相比于单纯的人物识别，这种融入生活场景中的“普适”物体，存在更多的噪声，具有更高的识别难度，也更具有现实意义。

（2）建模流程

TensorFlow 官网提供了对 CIFAR-10 的建模范例，其在 AlexNet 网络的基础上做了部分参数修改。本实例采用相同的网络结构和参数取值，以更加精简的方式构建模型，旨在让读者直观地学习深度神经网络的建模方式。

① 使用 cifar10.py 脚本读取 CIFAR-10 数据。该脚本由 TensorFlow 官网提供，封装了对 CIFAR-10 中图片的下载及预处理等过程。

② 构建 AlexNet 模型如图 5.40 所示。

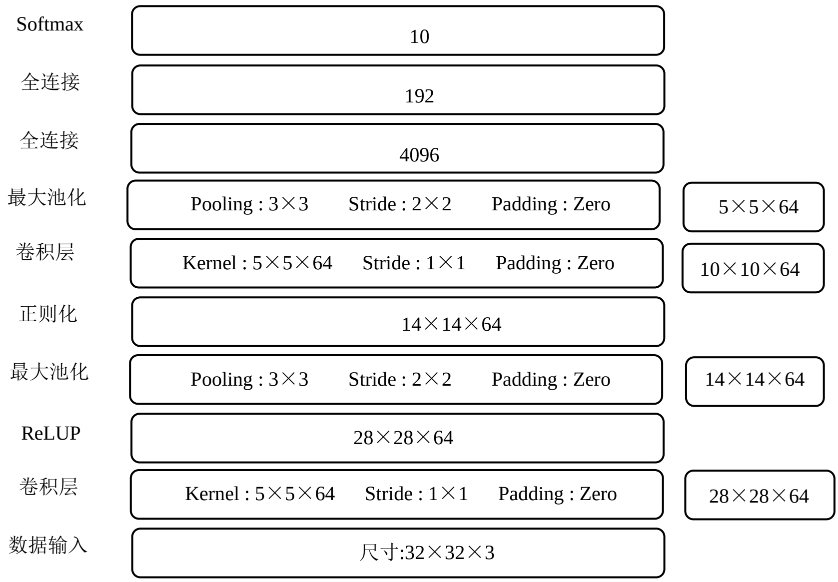


图 5.40 CIFAR-10 图像识别网络结构图

- 输入大小为 32 × 32，深度为 1 的图片；
- 执行卷积操作。卷积核大小：5 × 5；卷积核数量：64；横向和纵向步长 1 × 1；
- 采用零值填充。经过该卷积操作，生成大小为(32 − 5 + 1) × (32 − 5 + 1)，深度为 64 的图片；
- 执行 ReLU 操作，图片大小不改变；
- 执行最大池化。池化窗口 3 × 3; 横纵向步长 2 × 2。生成大小为(28 / 2) × (28 / 2)，深度为 64 的图片；
- 执行正则化操作；
- 执行卷积操作。卷积核大小：5 × 5；卷积核数量：64；横向和纵向步长 1 × 1；
- 采用零值填充。经过该卷积操作，生成大小为(14 − 5 + 1) × (14 − 5 + 1)，深度为 64 的图片；
- 执行最大池化。池化窗 3 × 3；横纵向步长 2 × 2。生成大小为(10/ 2) × (10 / 2)，深度为 64 的图片；
- 执行全连接操作。神经元数量：4096；
- 执行全连接操作。神经元数量：192；

- 执行全连接操作，神经元数量：10，执行 SoftMax 回归。

③ 使用 eval.py 脚本测试模型效果。首先加载 train.py 脚本训练出模型，之后读取测试数据集，进行准确率验证。

（3）建模代码

程序代码由 alexnet.py、train.py、cifar10.py、cifar10_input.py、eval.py 五个文件构成。其中 cifar10.py 和 cifar10_input.py 从 TensorFlow 官网的“CIFAR-10 建模范例”中下载，分别用于读取训练数据集和测试数据集；train.py 包含了程序启动入口函数；alexnet.py 包含构建网络的全部代码；eval.py 用于读取序列化在文件中的训练模型，并使用测试数据集测试。代码组织结构如图 5.41 所示。

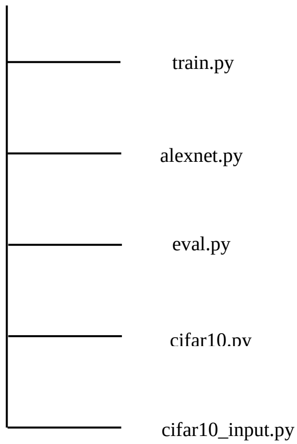


图 5.41　CIFAR-10 识别代码目录结构

train.py

```
#!usr/bin/python
# -*- coding:utf-8 -*-
import cifar10
import tensorflow as tf
import os
import math
import time
import alexnet
```

```
    # cifar-10 数据集
    cifar10.maybe_download_and_extract()
    images, labels = cifar10.distorted_inputs()

    # 构建模型
    pred = alexnet.alexnet(images)

    # 损失函数
    sparse_labels = tf.reshape(labels, [128, 1])
    indices = tf.reshape(tf.range(0,128), [128, 1])
    concated = tf.concat(1, [indices, sparse_labels])
    denseLabels = tf.sparse_to_dense(concated, [128, 10], 1.0, 0.0)
    cost                                                            =
tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(pred,
tf.cast(denseLabels, tf.float32)))

    # 最优化
    global_step = tf.Variable(0, trainable=False)
    lr = tf.train.exponential_decay(0.1, global_step, int(50000 / 128
* 350), 0.1, staircase=True)
    train   =   tf.train.GradientDescentOptimizer(0.1).minimize(cost,
global_step=global_step)

    # 准确度评估
    correct_pred = tf.equal(tf.argmax(pred, 1), tf.argmax(denseLabels,
1))
    accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))

    # 保存模型
    saver = tf.train.Saver()

    # 初始化所有的变量
    sess = tf.Session()
    init = tf.initialize_all_variables()
    sess.run(init)

    # 开始 queuerunners，异步读取数据
    tf.train.start_queue_runners(sess=sess)

    # 开启一个训练
    forstepinxrange(10000):
       _, loss = sess.run([train, cost])
    ifstep % 10 == 0:
          # 计算精度
    acc = sess.run(accuracy)
```

```
    print "Step = " + str(step) + ", BatchLoss = " + "{:.6f}".format(loss)
+ ", TrainingAccuracy= " + "{:.5f}".format(acc)
    ifstep % 1000 == 0:
    # 保存模型
    checkpoint_path = os.path.join("/tmp/cifar10_train", 'model.ckpt')
    saver.save(sess, checkpoint_path)
```

alexnet.py

```
    #!usr/bin/python
    # -*- coding:utf-8 -*-
    import cifar10
    import tensorflow as tf
    import os
    import math
    import time
    # cnnNet
    defcnnNet(images):
    # 第一层 : 卷积 (核大小 :5 * 5, 深度 : 3, 核数量:64)
    conv1Weights  =  tf.Variable(tf.random_normal([5,  5,  3,  64],
stddev=1e-4))
    conv1Biases  =tf.Variable(tf.random_normal([64]))
    conv1   =   tf.nn.bias_add(tf.nn.conv2d(images,   conv1Weights,
strides=[1, 1, 1, 1], padding='SAME'), conv1Biases)
    relu1 = tf.nn.relu(conv1)

    # 第二层 : 池化
    pool1 = tf.nn.max_pool(conv1, ksize=[1, 3, 3, 1], strides=[1, 2, 2,
1], padding='SAME')

    # 第三层 : 正则化
    norm1 = tf.nn.lrn(pool1, 4, bias=1.0, alpha=0.001 / 9.0, beta=0.75)

    # 第四层 : 卷积
    conv2Weights  =  tf.Variable(tf.random_normal([5,  5,  64,  64],
stddev=1e-4))
    conv2Biases  =tf.Variable(tf.random_normal([64]))
    conv2 = tf.nn.bias_add(tf.nn.conv2d(norm1, conv2Weights, strides =
[1, 1, 1, 1], padding='SAME'), conv2Biases)
    relu2 = tf.nn.relu(conv2)

    # 第五层 : 正则化
    norm2 = tf.nn.lrn(relu2, 4, bias=1.0, alpha=0.001 / 9.0, beta=0.75)
```

```
    # 第六层 : 池化
    pool2 = tf.nn.max_pool(conv1, ksize=[1, 3, 3, 1], strides=[1, 2, 2,
1], padding='SAME')

    # 第七层 : 全连接
    poolShape = pool2.get_shape().as_list()
    shapeNum  =poolShape[1] * poolShape[2] * poolShape[3]
    reshaped = tf.reshape(pool2, [-1, shapeNum])
    ip1Weights  =  tf.Variable(tf.random_normal([shapeNum,  64  *  64],
stddev=1e-4))
    ip1Biases  = tf.Variable(tf.random_normal([64 * 64]))
    ip1 = tf.matmul(reshaped, ip1Weights) + ip1Biases
    relu3 = tf.nn.relu(ip1)

    # 第八层 ： 全连接
    ip2Weights = tf.Variable(tf.random_normal([64 * 64, 192],
stddev=1e-4))
    ip2Biases = tf.Variable(tf.random_normal([192]))
    ip2 = tf.matmul(relu3, ip2Weights) + ip2Biases

    # 第九层 :softmax
    softmaxWeights = tf.Variable(tf.random_normal([192, 10],
stddev=1e-4))
    softmaxBiases = tf.Variable(tf.random_normal([10]))
    softmaxLinear = tf.add(tf.matmul(ip2, softmaxWeights),
softmaxBiases)
    returnsoftmaxLinear
```

eval.py

```
    #!usr/bin/python
    # -*- coding:utf-8 -*-
    import cifar10
    import tensorflow as tf
    import os
    import math
    import time
    import alexnet
    import numpy as np
    import date time

    # 读取测试数据
    images, labels = cifar10.inputs(eval_data = "test")
    logits = alexnet.alexnet(images)
```

```
    # 加载模型
    ckpt = tf.train.get_checkpoint_state("/tmp/cifar10_train")
    saver = tf.train.Saver()

    # 统计 t 准确率
    topK = tf.nn.in_top_k(logits, labels, 1)
    withtf.Session() assess:
    saver.restore(sess, ckpt.model_checkpoint_path)
    coord = tf.train.Coordinator()
    try:
    threads = []
    printtf.GraphKeys.QUEUE_RUNNERS
    forqrintf.get_collection(tf.GraphKeys.QUEUE_RUNNERS):
    threads.extend(qr.create_threads(sess, coord=coord, daemon=True,
start=True))
    num_iter = int(math.ceil(10000 / 128))
    true_count = 0
    total_sample_count = num_iter * 128
    step = 0
    whilestep<num_iterandnotcoord.should_stop():
    predictions = sess.run([topK])
    true_count += np.sum(predictions)
    printtrue_count
    step += 1

    # Computeprecision @ 1.
    precision = true_count / float(total_sample_count)
    print('%s: precision  = %.3f' % (datetime.datetime.now(),
precision))

    exceptExceptionase:  # pylint: disable=broad-except
    coord.request_stop(e)

    print "finished!"
    coord.request_stop()
    coord.join(threads, stop_grace_period_secs=10)
```

由图 5.42 可知，经过 5000 次训练，该模型在 CIFAR-10 的训练集上，识别准确率达到 64.00%，在测试集上达到 65.37%。如果增加模型的训练时长，会进一步提升效果。

运行结果如下。

```
[root@localhost alexnet]# python train.py
Filling queue with 20000 CIFAR images before starting to train. This will take a few minutes.
I tensorflow/core/common_runtime/local_device.cc:25] Local device intra op parallelism threads: 1
I tensorflow/core/common_runtime/local_session.cc:45] Local session inter op parallelism threads: 1
Step = 500, Batch Loss = 0.852723, Train Accuracy= 0.48667
Step = 1000, Batch Loss = 0.255415, Train Accuracy= 0.61333
Step = 1500, Batch Loss = 0.220778, Train Accuracy= 0.62667
Step = 2000, Batch Loss = 0.257257, Train Accuracy= 0.61333
Step = 2500, Batch Loss = 0.212687, Train Accuracy= 0.62000
Step = 3000, Batch Loss = 0.161957, Train Accuracy= 0.64000
Step = 3500, Batch Loss = 0.209988, Train Accuracy= 0.63333
Step = 4000, Batch Loss = 0.063564, Train Accuracy= 0.65333
Step = 4500, Batch Loss = 0.120610, Train Accuracy= 0.64000
Optimization Finished!
Test Accuracy: 0.653646
```

图 5.42　CIFAR-10 数据集识别结果

6 大数据分析

6.1 常用组件介绍

“工欲善其事，必先利其器”，具有特定功能的可复用组件正是计算机领域中的利器。在大数据的浪潮下，许多用于处理大数据的组件应运而生，分别应用在“数据传输”“数据存储”“数据计算”以及“数据展示”的环节中。本章将着重介绍一些常用组件的内部原理以及使用方式，并讲述在大数据处理领域中的一些通用架构模式。

6.1.1 数据传输

数据传输是数据处理至关重要的一步，数据工程师需要使用高效的传输方式，把分散在不同机房、不同地域的海量数据汇集在一起，才能进一步实现数据的处理。Kafka、Logstash 和 Flume 是大数据传输中三种典型的组件，本节分别介绍其架构原理以及使用方式，旨在让读者对大数据领域如何传输数据有一定的了解。

1．Kafka

Kafka 是用 Scala 编写的分布式消息处理平台，最初由 LinkedIn 公司开发，后成为 Apache 项目的一部分。与 Kafka 类似的常用消息队列包括 RabbitMQ、ZeroMQ、ActiveMQ 等，它的架构如图 6.1 所示。

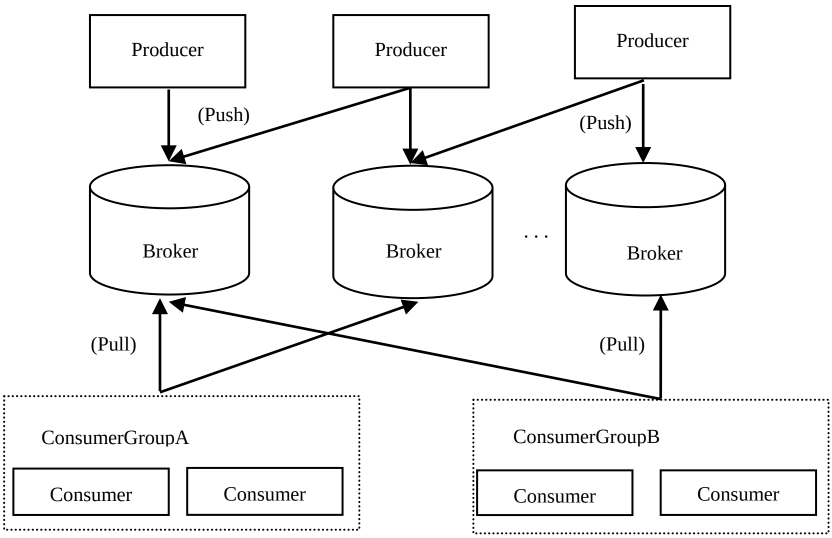


图 6.1　Kafka 架构图

Producer：消息的生产者，主要负责向 Broker 发送消息。Kafka 的负载均衡方式由 Producer 自身决定，这就意味着由 Producer 按照 Round Robin、随机的方式或者其他方式向 Broker 集群发送消息。

Broker：负责接受与存储 Producer 发送来的消息。一个 Broker 就是一个 Kafka Server，一个集群由 N 个 Broker 组成。

Consumer：消息的消费者，负责从 Broker 拉取数据。

ConsumerGroup：由 N 个 Consumer 组成的组。同一个 Group 的 Consumer 接受同一 Topic 的数据。Topic 是 Broker 在组织与处理消息时用到的概念。Broker 内部处理 Topic 的方式如图 6.2 所示。

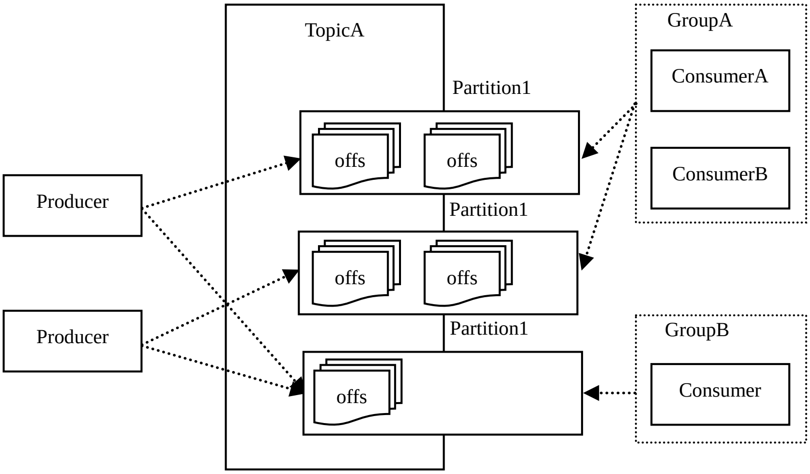


图 6.2　Topic 组织结构图

Topic：逻辑上的概念，可以理解成消息队列。Producer 向指定的 Topic 中发送数据，Consumer 消费来自指定 Topic 的数据。

Partition：一个 Topic 由多个 Partition 组成，分布在多个 Broker 中，实现负载均衡。同时，其在 Broker 集群中会保存多个副本，是 Kafka 集群实现可靠性保障的最小单位。如图 6.3 所示，单个 Partition 中的消息根据先后顺序不断追加。

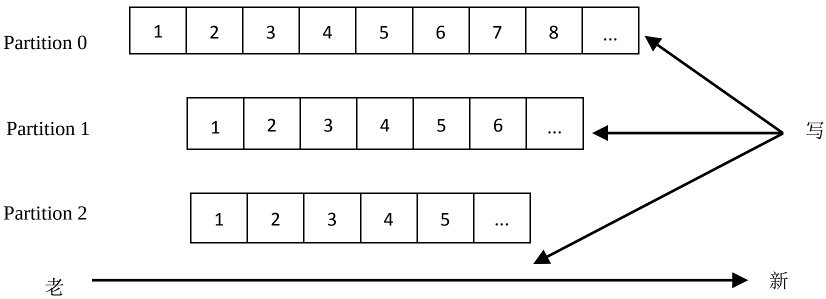


图 6.3　Partition 数据写入

Offset：每个 Partition 在 Broker 机器中会分割成多个小的日志文件存储在磁盘上，这些日志文件按 Offset 的形式分割存储。

2. Logstash

Logstash是用JRuby编写的一种分布式日志收集框架，由Input、Filter和Output三个部分构成，分别负责日志的输入、格式化以及输出。与 Kafka 这种消息队列相比，它更像是消息的传送带，连接源头接收数据，在中间传输的过程中做格式化处理，Output 最终将消息以 Push 的方式推送到目的地。它的架构如图 6.4 所示。

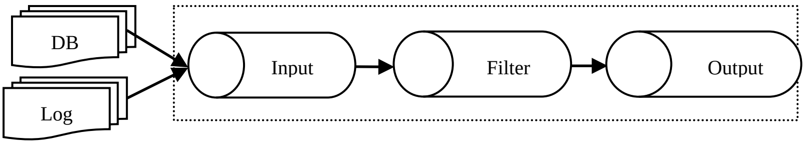


图 6.4 Logstash 架构图

Logstash 的各个功能模块通过配置的方式定义输入的监听端口、输出的目的地，以及格式化的方式。以下是来自官网的一个配置示例，可以让读者有直观的认知。

```
input{
beats{
port=>"5043"
}
}
filter{
grok{
match=>{"message"=>"%{COMBINEDAPACHELOG}"}
}
}
output{
stdout{codec=>rubydebug}
}
```

input 可以配置很多的数据接收插件，比如 Redis、filebeat 等；以上配置中 input 代码块表示添加 beats 的输入插件；filter 配置说明对日志消息格式化的方式；output 配置日志打印到控制台。这是 Logstash 配置的基本骨架，可参照官方文档定制修改。

3．Flume

同 Logstash 类似，Flume 是用 Java 编写的分布式实时日志收集框架，它的原始版本由 Cloudera 公司开发，称为 Flume OG。随着功能的不断扩展，核心组件设计的问题逐渐暴露，稳定性问题频发，迫使 Cloudera 对 Flume 实施大规模改动——重构后的 Flume 被称为 Flume NG，即 Flume Next Generation。它最终被纳入 Apache 旗下，改名为 Apache Flume，它的架构如图 6.5 所示。

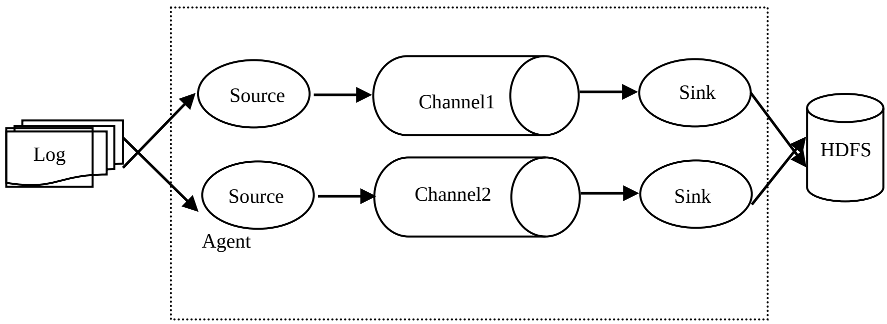


图 6.5　Flume 架构图

① Agent：每个 JVM 对应一个 Agent 进程，一个 Agent 中包含多个 Source、Channel、Sink。

② Source：用于搜集数据，并把数据传输到 Channel。

③ Channel：缓存 Source 发来的数据。当前有 MemoryChannel、JDBCChannel、FileChannel、CustomChannel 等可供配置。如果 Channel 中的数据选择存储在内存中，那么速度快，但易丢失；相反，如果存储在硬盘中，则可靠性更高。

④ Sink：从 Channel 中读取数据，可以向硬盘、HDFS、数据库等存储介质传输数据。Flume 本身支持 HDFS Sink、Thrift Sink 等类型。

Flume 与 Logstash 的定位基本一致，都是作为“传送带”把数据从一端运送到另一端。不同的是，前者强调数据的传输，但是基本不会对数据做字段提取等操作；后者在传输数据的同时，会依据配置做字段提取、解析等预处理工作，和 Flume 相比，Logstash 会耗费更多的系统资源、加重系统的负载。此外，Flume 的数据保存在 Channel 中，只有流转完成后，才会删除数据，从而进一步保证了数据的可靠性。

6.1.2　数据存储

集中存储是有效分析数据的前提条件，然而，如何存储 TB 级别甚至 PB 级别的数据，对数据工程师是一个巨大的挑战。大数据时代，为了解决数据集中存储的问题，各种组件层出不穷，根据应用领域的不同，本书选取 HDFS、Redis、HBase、Hive 以及 Elasticsearch 这 5 种典型的组件予以介绍。

1．HDFS

HDFS（Hadoop Distributed File System）用 Java 编写，是谷歌的 GFS 的一种开源实现。HDFS 可在廉价的机器上实现存储能力的横向扩展，且以多副本的方式实现数据的可靠性保障，它的架构如图 6.6 所示。

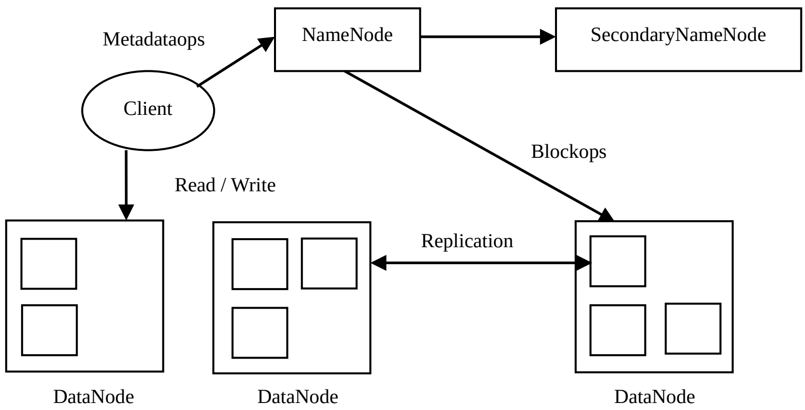


图 6.6　HDFS 架构图

由图 6.6 可知，HDFS 采用了 Master-Slave 的架构方式。其中 NameNode 和 SecondaryNameNode 分别为 Master 的主节点和备节点，复制管理文件的目录、文件、DataNode 以及 Block 的对应关系。

HDFS 适合大文件一次上传，多次读取；不支持对已上传数据块的随机读写。图中小方块代表了 Block，大文件会被分成多个小 Block 存储到多个 DataNode 中；同一个 Block 也会被复制到其他 DataNode 中实现数据的多备份。

文件的写流程如下。

① Client 上传大文件时，首先在本地文件系统建立临时存储区，当数据量积累到一个数据块大小时，再与 NameNode 会话，获取 DataNode 位置和 Block 的标识，进行写入。

② 当数据副本设置为 3 时，DataNode 使用流水式的复制方式实现数据的接收和备份。第一个 DataNode 一边接收数据，一边把数据发送给第二个 DataNode；第二个 DataNode 以相同的方式把数据传送给第三个 DataNode。当三个数据块全部写完则认为一个 Block 上传成功。

③ 当文件关闭时，如果临时存储区还有不满一个 Block 的数据量，也会一并上传至 HDFS，并告知 NameNode 写入结束。

④ NameNode 将文件操作提交给日志存储。

文件的读流程如下。

① 与 NameNode 会话，获取待读取文件的 DataNode 和 Block 信息。

② 选择“最近”的副本读取数据。

③ 读取完毕，发送 Close 给 NameNode，断开连接。

2. Redis

Redis 是用 ANSIC 编写的一种基于内存的 Key-Value 键值对数据库，类似于早期的 memcached，可以在内存数据库、缓存、消息代理等场景中使用。以下是 Redis 的常用数据结构及示例。

String 示例：

```
127.0.0.1:6379>SET "testKey"  "testValue"
OK
127.0.0.1:6379>GET "testKey"
"testValue"
```

Hash 示例：

```
127.0.0.1:6379>HSET "testField" "testKey" "testValue"
(integer) 1
127.0.0.1:6379>HGET "testField" "testKey"
"testValue"
127.0.0.1:6379>HGETALL "testField"
1) "testKey"
2) "testValue"
```

List 示例：

```
127.0.0.1:6379>LPUSH "testList" "1"
(integer) 1
127.0.0.1:6379>LPUSH "testList" "2"
(integer) 2
127.0.0.1:6379>LPUSH "testList" "3"
(integer) 3
127.0.0.1:6379>LRANGE "testList" 0 2
1) "3"
2) "2"
3) "1"
```

Set 示例：

```
127.0.0.1:6379>SADD "testSet" "1"
(integer) 1
127.0.0.1:6379>SADD "testSet" "2"
(integer) 1
127.0.0.1:6379>SADD "testSet" "3"
(integer) 1
127.0.0.1:6379>SADD "testSet" "3"
(integer) 0
127.0.0.1:6379>SMEMBERS  "testSet"
1) "1"
2) "2"
3) "3"
```

Redis 更高级的使用方式比如“SortedSet”“发布-订阅”等可参见官网。下面我们介绍 Redis 的其他特性。

Redis 支持用“日志快照”和“追加日志”的方式实现数据的持久化。“快照”主要是每隔一段时间把内存的数据完整地复制到磁盘；而“追加”则是对每一条

操作追加一条日志。使用日志快照的方式实现数据持久化时，两次快照具有一定的间隔时间，在该时间窗口内如果 Redis 发生故障，则会造成数据的丢失；采用追加日志的方式实现数据持久化时，由于实时记录每一条操作，不存在快照方式中的“时间窗口”，因此，可以降低数据丢失的风险，但是在故障恢复时，如果采用追加日志的方式实现持久化，工程师要使用工具逐条重放日志才能恢复数据，速度较慢。

Redis 支持主从复制，即把一台 Redis 服务器的数据复制到多台备份机器上，并且这种复制是异步的，不会影响主服务器的性能。这种主从机制可以用来实现读写分离，并在某些场景下替代数据的持久化。

Reids 虽然是单机服务器，但是可以采用一定方式组建 Redis 服务集群，即通过一致性哈希的方式将其组织起来，并最终在客户端实现分布式的应用。这需要使用者根据需求自行决定。

3．HBase

HBase 是用 Java 编写的分布式列式数据库，它仿照的是谷歌列式数据库 BigTable 的开源实现。HDFS 是分布式文件系统，HBase 则是在该文件系统的上层实现的数据库，它的架构如图 6.7 所示。

由图可知，一个 HMaster 管理多个 HRegionServer；一个 HRegionServer 管理多个 HRegion；一个 HRegion 管理多个 Store 模块；每个 Store 模块下又由 MemStore 和多个 StoreFile 构成，StoreFile 是对 HFile 的轻量级封装。

在说明 HBase 的读写流程之前，需要先介绍两张表-ROOT-和.META。其中.META 记录了用户表的 HRegionServer 信息；由于.META 表可能分裂到多个 HRegionSever 上存储，因此-ROOT-记录了.META 表的 Region 信息，-ROOT-只有一个 Region。-ROOT 和.META 共同构成了 HBase 的索引，如图 6.8 所示。

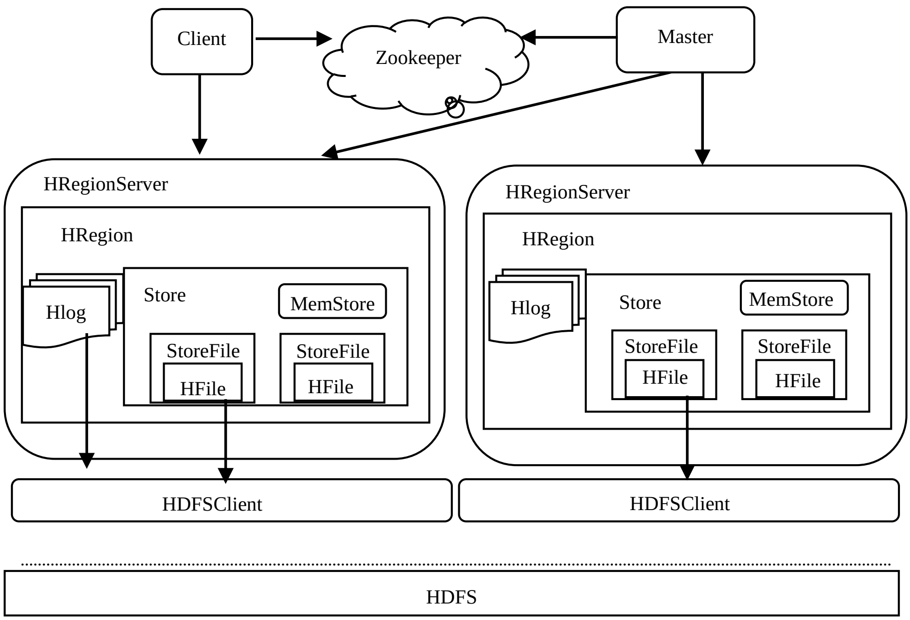


图 6.7　HBASE 架构图

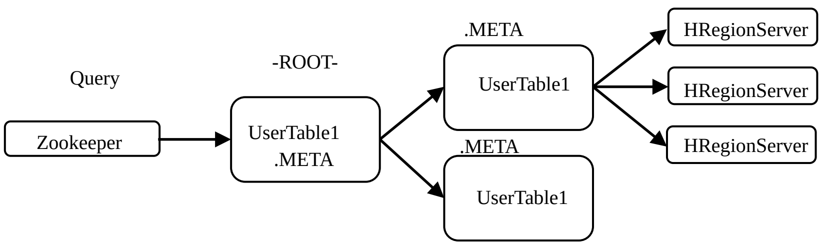


图 6.8　HBase 索引结构示意图

写入的流程如下。

① 每个表的数据都有唯一的 RowKey（行键）。

② 根据表名字和 RowKey，从-ROOT-和.META 中定位写入的 HRegionServer 的地址。

③ HRegionServer 根据配置决定是否写入 Hlog;之后把数据写入到 MemStore，如果其已满，则刷新成一个 StoreFile，即 HDFS 的一个文件。

HBase 使用 RowKey，根据-ROOT-和.META 表直接定位数据的位置，读取数

据。如果某个 RowKey 被查询过，它的相关信息会被缓存，后续针对这个 RowKey 的查询会非常之快。

HBase 中 RowKey 决定写入的 HRegionServer 地址，因此 RowKey 的设计会决定写入、读取的负载程度。最初一个 Table 只有一个 HRegion，随着数据的增加，HMaster 会对其进行 Split 操作，将其一分为二；同时下线父 HRegion，而新的子 HRegion 上线；子 HRegion 可能被分配到不同的 HRegionServer 中以实现负载均衡。

HBase 采用 LSM 作为数据存储引擎，大致的思路是数据按顺序写入，且更新数据就相当于写入一份新数据，并不更改原数据。当 HFile 的个数超过一定的阈值时，就会触发 Compact 操作，对所有数据版本进行合并和删除。

HBase 的数据存储在 HDFS 中，实现了数据的多副本存储；同时引入 Hlog 降低了写入 MemStrore 过程中数据丢失的风险。同一行数据同一时间只有一个 HRegion 提供访问，因此 HBase 支持行级锁，且实现了数据的强一致性；如果某一 HRegion 宕机，则需要一段时间用日志更新其他 HRegion 后才能继续提供服务。

4．Hive

Hive 是用 Java 编写的、建立在 Hadoop 之上的分布式数据仓库。与 HBase 数据库相比，两者的定位截然不同：数据仓库针对大量的历史数据，提供非实时的联机分析操作；而数据库针对量级较小的业务数据，提供实时的事务处理操作。Hive 开发了自身的 SQL 语言，称为 HQL。Hive 的数据存储在 HDFS 上，HQL 语句经过解析生成 MapReduce 任务，依托 Hadoop 的 MapReduce 计算框架进行数据的处理。它的架构如图 6.9 所示。

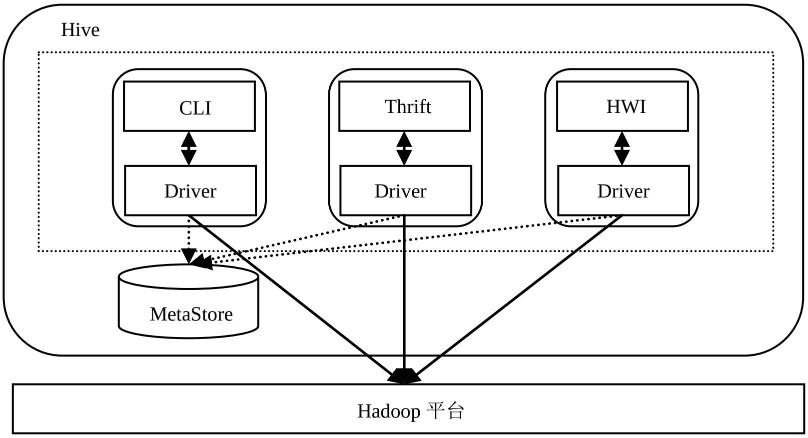


图 6.9　Hive 架构图

Hive 提供 CLI、Thrift 以及 HWI 三种对外服务。其中 CLI 是命令行工具;Thrift 是 RPC 服务；HWI 为 Web 接口，可以通过浏览器访问 Hive。客户端通过上述三种服务把 HQL 提交到 Hive，Hive 负责 HQL 的解析、编译、生成，并提交 MR 任务到 Hadoop 平台，这个流程如图 6.10 所示。

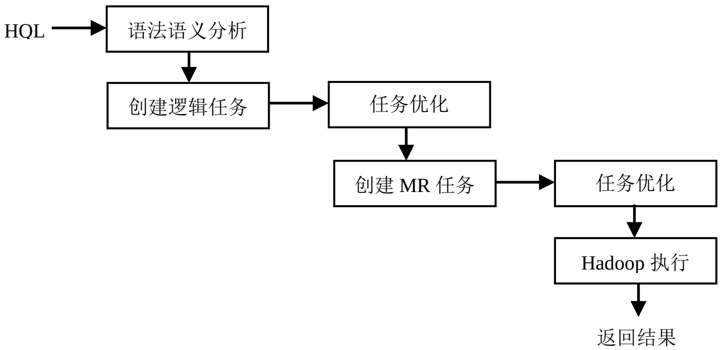


图 6.10　HQL 执行流程

由图 6.10 可知，Hive 依赖 Hadoop 做数据运算，而 Hadoop 适用于海量数据的非实时处理，因此 Hive 的速度是它的短板。Shark、Tez、SparkSQL 等与 Hive 具有相同或相似功能的组件，在计算速度上都优于 Hive；但是 Hive 的稳定性、强大功能及与 Hadoop 的完美融合，使它成为首选的分布式数据仓库。

5．Elasticsearch

Elasticsearch 是以 Apache Lucene 为核心打造的分布式全文搜索引擎。由于它还实现了分布式文件存储和部分实时分析的功能，有时候也把它当成文档数据库来使用。

Lucene 是一个开源的，用于文档索引和搜索的高性能 Java 库，其核心只有一个 jar 文件[32]。我们举例说明它的使用。

假设有三个文档分别是 No1: {"Iamalawyer"}、No2:{"Tomorrowisagoodday"}、No3:{"Partyisfulloffun"}，这些文档存储在数据文件中。Lucene 库的接口提供了这样的功能：指定数据文件的路径及相关参数，就可以对文档创建全文索引；当调用 search("lawyer")函数搜索时，接口可以及时返回文档 No1。当然，Lucene 库还包含文件过滤、排序等辅助功能。

Elasticsearch 以 Lucene 框架为核心，构造了分布式的全文搜索引擎。它对外提供两种访问方式：Java API 交互和基于 HTTP 协议的 Restful 接口。下面我们给出 Restful 接口的一个例子，以使读者有直观的认识。

```
#添加索引
PUT /nginx/log/1
{
   "date" : "2017/02/13",
   "text" : "www.xxx.com, nothaveasubpage"
}
#获取文档
GET /nginx/log/1
{
   "match":{
   "text" : "haveasubpage"
   }
}
#返回结果
{
   "_index" : "nginx",
   "_type"  :  "log",
   "_id"    : "1",
   "found"  : true,
```

```
    "_source" : {
        "date" : "2017/02/13",
        "text" : "www.xxx.com, nothaveasubpage"
    }
}
```

我们以 MySQL 为类比，阐述“返回结果”中的_index, _type 以及相关的几个重要概念，如图 6.11 所示。

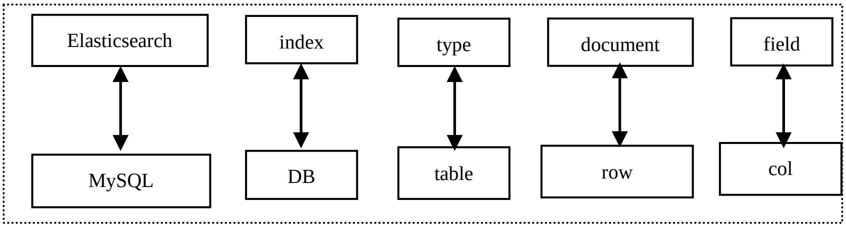


图 6.11　Elasticsearch 与 MySQL 类比图

如果我们把 Elasticsearch 看成是文档数据库，那么它的 index 与 MySQL 的 DB 实例其实是一个概念。请注意这里的 index 和 MySQL 中的 index（索引）有很大的区别；type 表示同一 scheme 存储的数据，此处的 scheme 就是 PUT 数据时的 JSON 格式；document 泛指一个文档，与 MySQL 的 row 的概念类似；一个 document 可以划分成多个 field。Elasticsearch 的集群架构如图 6.12 所示。

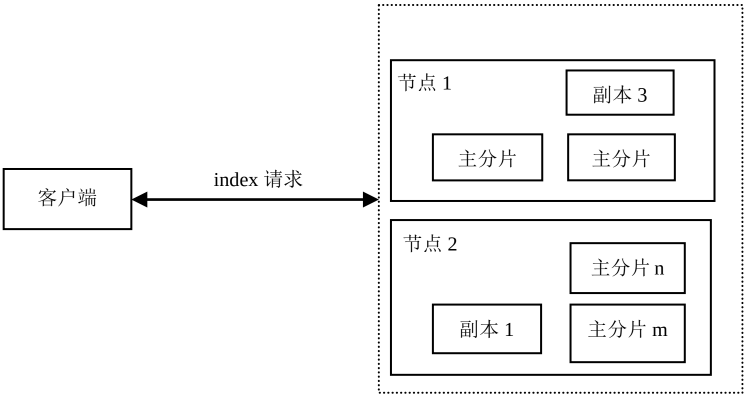


图 6.12　Elasticsearch 集群架构图

Elasticsearch 会把 index 进行切片，负载均衡到集群中多个计算节点中；同时，

会备份分片作为副本，这些副本分散在集群中，以实现数据存储的可靠性。

Elasticsearch 的任意节点都对外提供服务，不存在 Master-Slave 这种方式。它的工作流程如下。

① 客户端请求发送给 Elasticsearch 集群中任意一个节点 NodeA，该节点根据请求携带的 ID，计算出数据应存储在分片 Shard0 上。

② 根据集群信息，获取该分片 Shard0 对应节点 NodeC 的物理地址。

③ 所有到 NodeA 的请求，将被转发到 NodeC 的 Shard0 上进行读写操作。

6.1.3 数据计算

当前的高性能 PC 机、中型机等机器在处理海量数据时，其计算能力、内存容量等指标都远远无法达到要求。在大数据时代，工程师采用廉价的 PC 机组成分布式集群，以集群协作的方式完成海量数据的处理，从而解决单台机器在计算与存储上的瓶颈。本节主要介绍 Hadoop、Storm 以及 Spark 三种常用的分布式计算组件，其中 Hadoop 是对非实时数据做批量处理的组件；Storm 和 Spark 是针对实时数据做流式处理的组件。

1. Hadoop

Hadoop 是受 Google Lab 开发的 MapReduce 和 Google File System(GFS) 的启发而实现的开源大数据处理平台。Hadoop 的核心由 HDFS 分布式文件系统和 MapReduce 编程框架组成。前者已经在前述章节中有过介绍，它为海量数据提供了存储；后者则用于对海量数据的计算，是本节要着重介绍的内容。

MapReduce 是一种通用的编程模型，下面对它做简单介绍，它的工作流程如图 6.13 所示。

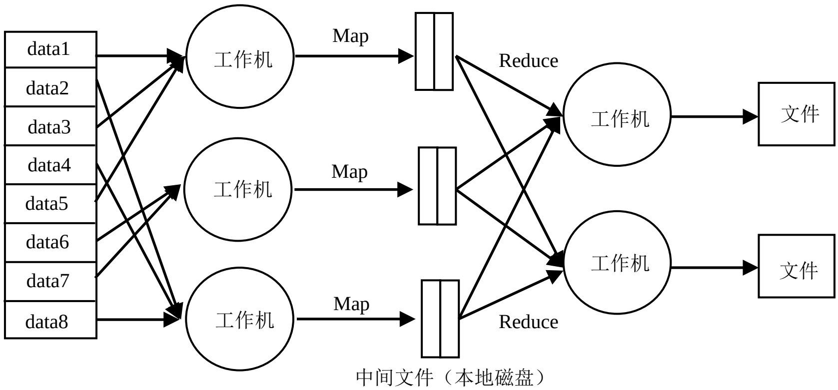


图 6.13　MapReduce 工作流程

我们以字母统计为例说明上述流程。假设有文件内容为“Iamapanda, andIamfromChina”。首先，把大文件分割成 data 数据块；其次，把 data 发送到各个工作机；此时，工作机解析内容，形成 Key-Value 键值对数据。本例中形成的数据为<I, 1>, <I, 1>, <am, 1>, <am, 1>, <a, 1>, <panda, 1>, <and, 1>, <from, 1>, <China, 1>，这些数据保存在中间文件中，Map 阶段结束。之后，根据 Key 值路由，把相同 Key 值的键值对路由到同一台工作机，并在工作机上实现单词计数。本例中计数结果<I, 2>, <am, 2>, <a, 1>, <panda, 1>, <and, 1>, <from, 1>, <China, 1>。最后，各 Reduce 工作机把结果写入文件，Reduce 阶段结束。

Hadoop 平台上通过 JobTracker 和 TaskTracker 协调调度，实现 MapReduce 的运行，其工作机制可以用图 6.14 说明。

由图 6.14 可知，JobTracker 负责任务调度，而 TaskTracker 负责任务的执行；同时，需要处理的数据存储在 HDFS 中，TaskTracker 根据 MR 程序读取并处理数据。

以上对 Hadoop 的介绍依据的是 Hadoop1.0（第一代 Hadoop）的整体框架，当前 Hadoop2.0（第二代 Hadoop）引入了 YARN 作为其资源调度的方式，架构与 1.0 略有不同，但依然采用 MR 的计算模型[33]。

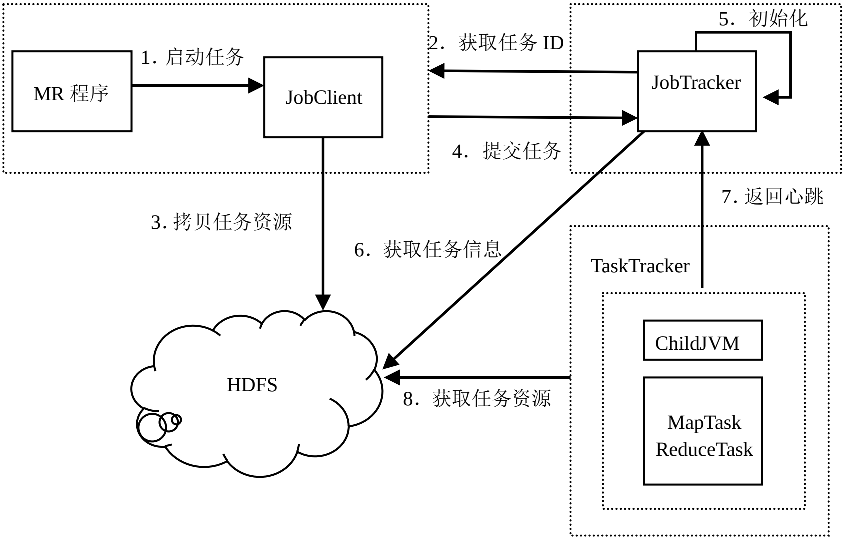


图 6.14　MapReduce 实现机制

2．Storm

Storm 是用 Clojure 语言编写的分布式实时流处理系统。Hadoop 平台执行批处理操作，数据处理的延迟较高；而进入 Storm 的数据则像水流一样源源不断流入，并对其做实时处理。Storm 集群架构如图 6.15 所示。

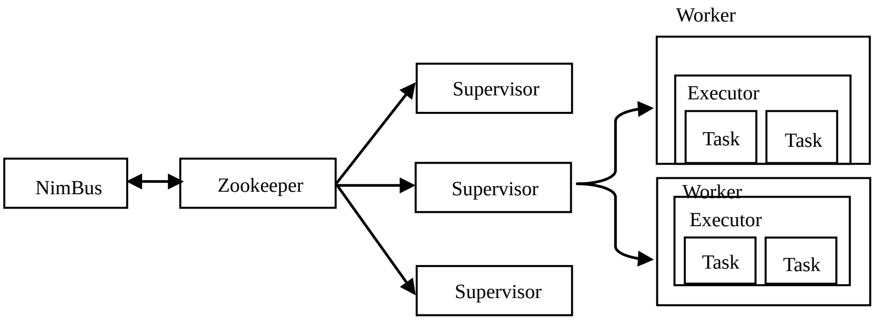


图 6.15　Storm 集群架构

Nimbus 与 Hadoop 中 JobTracker 的功能类似，负责资源的管理和任务的调度。从 Zookeeper 中读取各节点信息，协调整个集群的运行。

Supervisor 与 Hadoop 中 TaskTracker 的功能类似，负责接受任务，负责自身 Worker 进程的创建和任务的执行。

Worker 是机器上具体的运行进程，Executor 是该进程中的线程。一个 Executor 可以执行多个 Task。在该集群架构的方式下，Storm 实现了如图 6.16 所示的计算模型。

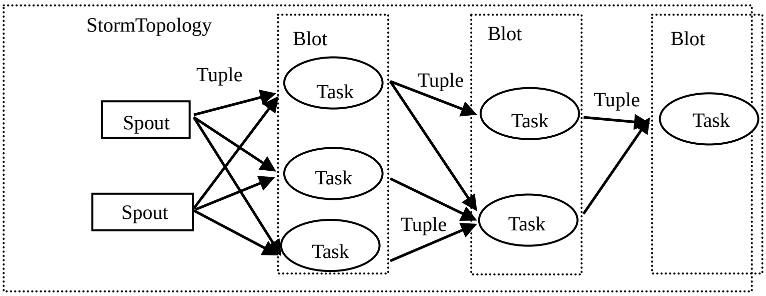


图 6.16 Storm 计算模型

Spout 是数据的入口，负责接受推送的数据，或者主动拉取数据。同时，把接收的数据转换为 Tuple 对象发送到 Blot 中处理。数据从 Spout 进入，封装成 Tuple，传输到第一层的某个 Blot 中，该 Blot 处理完成后，路由到第二层的某个 Blot 中，依此类推直到最后一组 Blot 处理结束。

Blot 是 Storm 实际的数据处理单元，接受 Spout 或者上一级 Blot 传输的数据并处理。根据并发度的设置，Blot 会分散到集群的一台或多台集群上并发执行，从而有效利用集群的计算能力，提高数据处理的实时性。这和在单台机器上多线程处理有相似之处。

Tuple 是一个或多个包含键值对的列表。数据会封装成 Tuple 对象在 Spout 与 Blot 之间传输。Storm 支持 7 种路由策略，分别为

- Shuffle 分组，Tuple 随机分散传输到后续的多个 Task 中；
- Fields 分组，根据指定 field 来做哈希，相同的哈希值传输到同一个 Task；
- All 分组，广播式地发送，把所有的 Tuple 发送到所有的 Task 中；
- Global 分组，把所有的 Tuple 发送到一个 Task 中；

- None 分组，也就是不关心如何路由，目前等同于 shuffle 分组；
- Direct 分组，是一种特殊的分组，需要手动指定 Task；
- Localorshuffle 分组，如果目标 Blot 中的 Task 和产生数据的 Task 在同一个 Worker 中，就执行线程间的内部通信，否则等同于 shuffle 分组。

3．Spark

Spark 是用 Scala 语言编写的分布式数据处理平台。Spark 的核心数据处理引擎依然是运行 MapReduce 计算框架，并且围绕该引擎衍生出多种数据处理组件，共同打造了轻量级的数据处理生态圈，如图 6.17 所示。

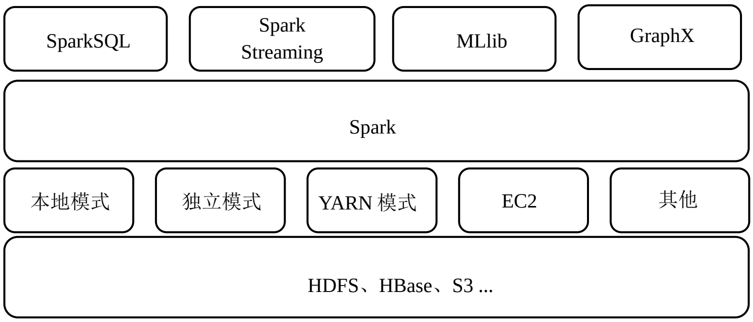


图 6.17　围绕 Spark 构建的数据处理组件

Spark 数据引擎是各组件库的核心。Spark 与 Hadoop 的计算框架都是基于 MapReduce 模型的，Spark 自身不包含类似 HDFS 的文件系统模块，而是借助外部的平台如 HDFS、HBase 等存取数据。Spark 在执行 MapReduce 的过程中做了重要的优化：第一，计算的中间数据不写磁盘，全部在内存中执行（可以设置对磁盘的依赖）；第二，支持任务的迭代。Hadoop 任务必须依照 Map→Reduce 成对执行，然而 Spark 可以依据任务的 DAG 图，按照 Map→Map→Reduce 等任意方式执行。这两点改进极大缩短了任务时延。

如图 6.18 所示为 Spark 的工作流程。RDD 是 Spark 的重要概念，代表了数据集和操作的结合。数据集来自内部或者外部，操作包含 map，group，reduce 等。我们下面给出一个 RDD 的示例。

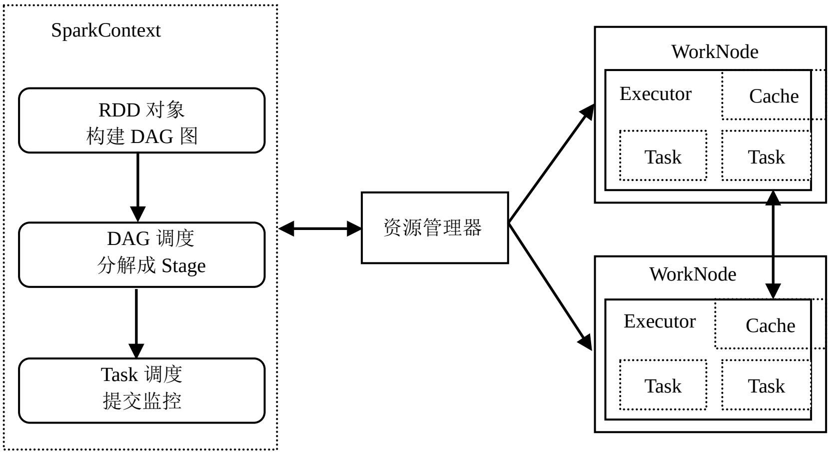


图 6.18　Spark 任务提交及集群架构

```
val lines = sc.textFile("data.txt")
val lineLengths = lines.map(s =>s.length)
val totalLength = lineLengths.reduce((a, b) =>a + b)
```

lineLengths 就代表数据集 lines 和操作 map 组成的 RDD。一个 RDD 又可以分多个 Task 执行，按照其执行的顺序组成 DAG 图。后续 RDD 的执行依赖先前 RDD 的执行，因此这种依赖关系又可以划分为 Stage，如图 6.19 所示，直观说明了 DAG、Stage、RDD 以及 Task 的概念。

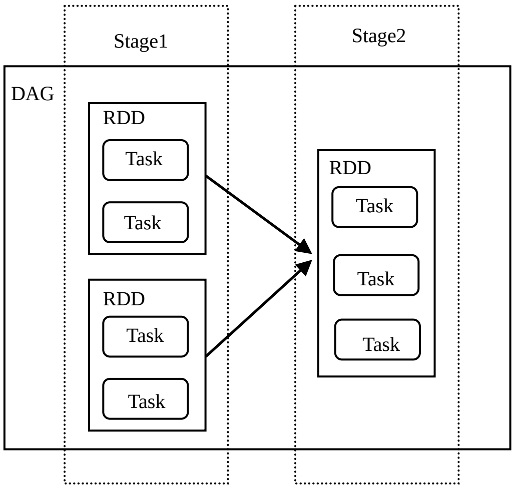


图 6.19　DAG、Stage、RDD、Task 逻辑关系图

Spark Streaming 是基于 Spark 核心处理引擎实现的高吞吐与低延迟的分布式流处理系统。与 Storm 相比，两者在功能上是一致的，都实现了数据流的实时处理；Storm 的延迟在亚秒级别，而 Spark Streaming 是在秒级别，主要因为前者对数据的处理就像水流一样，来一条数据则处理一条，而后者是不断进行小批量处理，只有在某些苛刻的场景下才能对比出这两种方式的优劣。Spark Streaming 数据处理流程如图 6.20 所示。

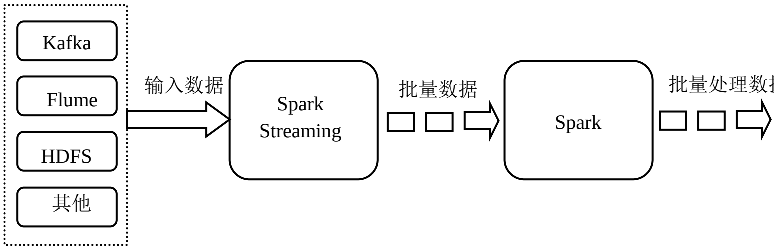


图 6.20　Spark Streaming 数据处理流程

SparkSQL 是分布式 SQL 查询引擎，与 Hive 类似，并对 Hive 提供支持。Hive 基于 Hadoop 的 MapReduce 实现查询，而 SparkSQL 则是基于 Spark 引擎，因此查询速度更快。但是，SparkSQL 需要更多的内存，在实际应用中其功能的丰富性和稳定性却不如 Hive。不过随着系统的不断演化，SparkSQL 将逐渐取代 Hive，成为分布式 SQL 查询引擎的佼佼者。

MLlib 是 Spark 封装的一些常用的机器学习算法相关库。基于 RDD 的方式实现了二元分类、回归、系统过滤等一些算法。GraphX 主要对并行图计算提供支持，开发并实现了一些和图像相关新的 Spark API。

6.1.4　数据展示

如果数据能以图表，折线等可视化的方式表现出来，那么数据工程师可以更加直观便捷地展示、发掘数据的信息，因此，数据可视化是数据处理的重要一环，它也有如 Spagobi、Kibana 等一些典型的组件可供复用。

1. Spagobi

Spagobi 是一款由 Engineering Group 实验室管理,完全开源的商业智能套件。2015 年 9 月 20 日，Engineering Group 发布 Spagobi 的第一个版本，并着力于将它打造成具有多种分析能力的商业智能平台。至今，该平台已经集成了自助 BI、Ad-hoc 查询、实时 BI、数据挖掘等多种强大数据处理与数据交互的能力。

如图 6.21 所示，通过官方提供的“All-in-one”包部署并启动服务后，即可在浏览器中通过 URL 访问 Spagobi 的交互界面。该图概括说明了平台所具有的能力。平台具体支持如下数据的交互与操作：Charts、Reports、Olapcubes、Cockpits、Map、Monitoringconsole、Networkanalysis、KpiModel、Qbe、SmartFilter、Dossier、Whatif、DataMining。我们下面给出一些具体的实例直观地说明 Spagobi 的使用。

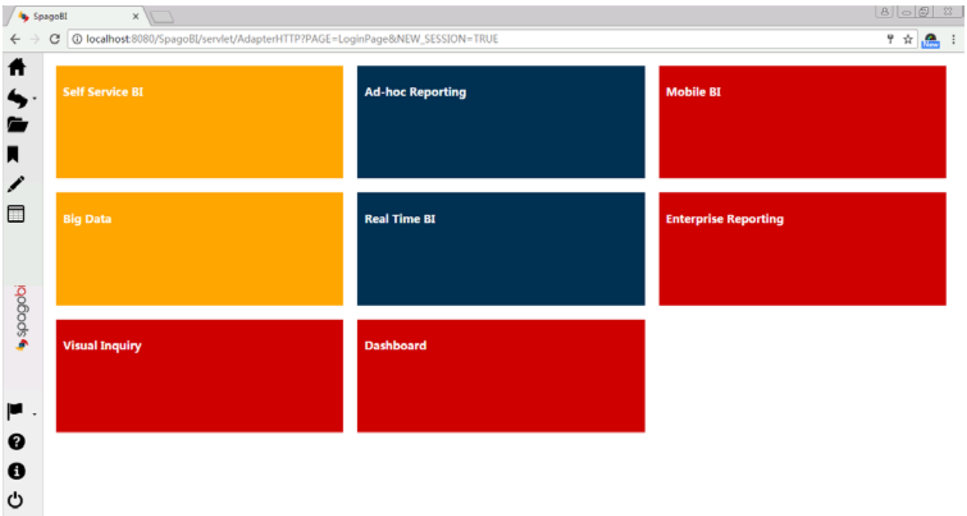


图 6.21 Spagobi 主页示意图

如图 6.22 所示是官网提供的“图表”样例。其他样例还包括：折线图、柱状图、饼状图等，此处不再赘述。图 6.22 的右侧，可以选定时间参数，从而指定某段时间内的数据。



图 6.22　Spagobi 的图表样例

如图 6.23 所示是官网提供的“报表”样例。相比于图表，报表融入了图标、文本等更多的元素。同样，报表也可以指定相应的参数来选择数据。

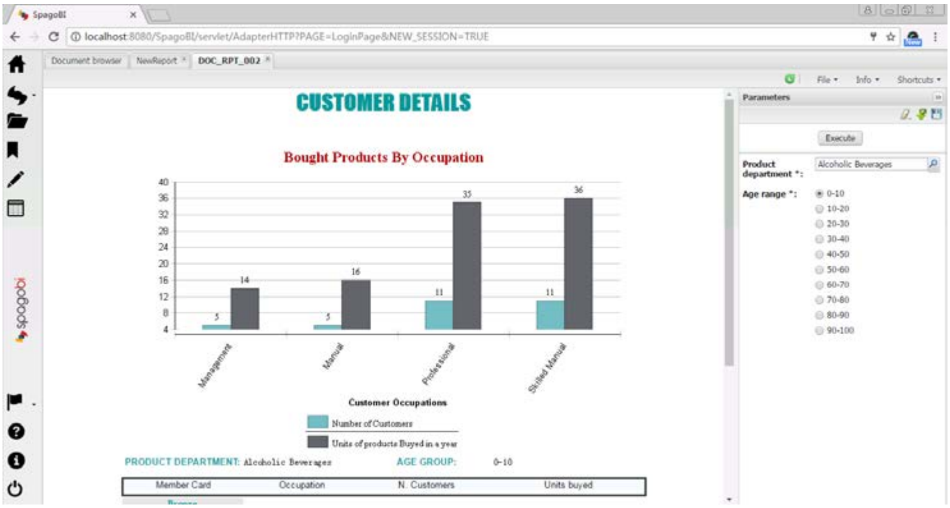


图 6.23　Spagobi 的报表样例

如图 6.24 所示，用户可以通过手动点击“+”和“–”按钮来实现“上卷下钻”的操作。这种方式极大方便了终端用户，使他们可以依据具体的场景选择合适的维度进行查询操作。

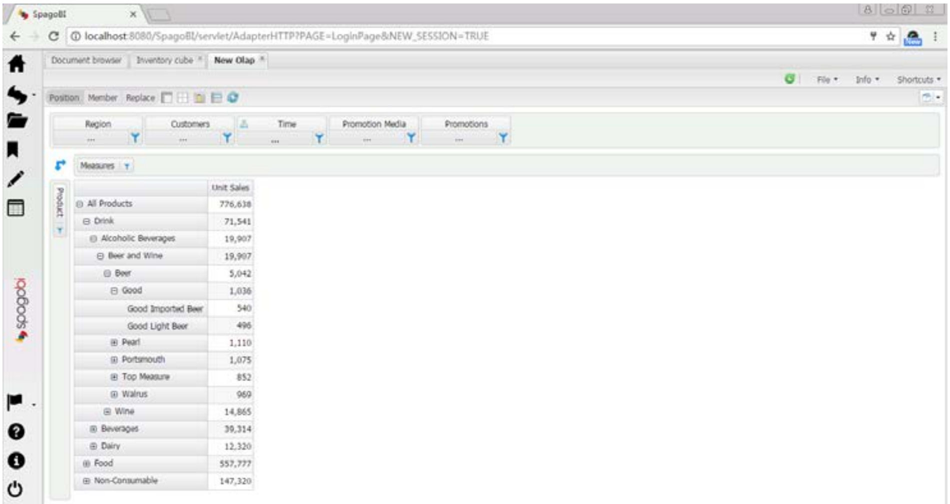


图 6.24　Spagobi 的 OLAP 分析样例

如图 6.25 所示，Spagobi 将数据具有的地理位置信息，与具体的地图模型关联，以极其形象的方式展示不同地域、不同纬度的数据信息。

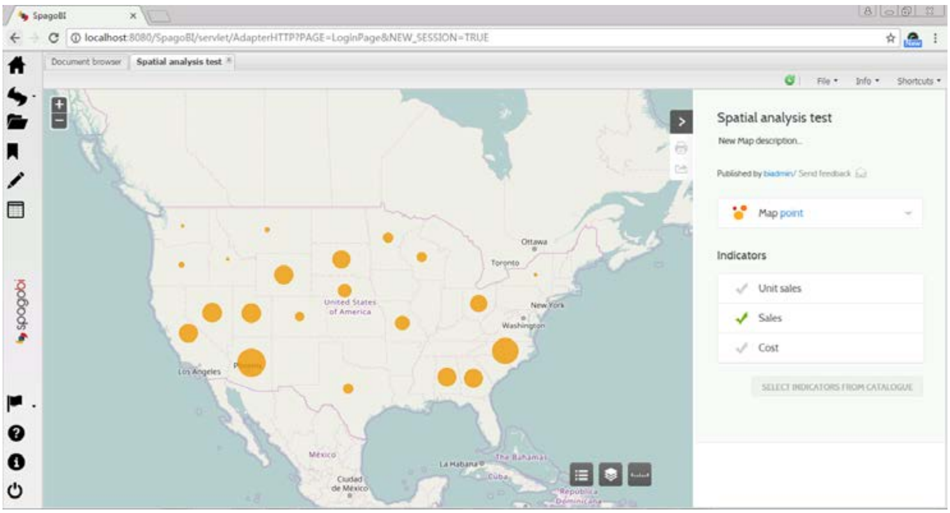


图 6.25　Spagobi 的 Map 分析样例

如图 6.26 所示，针对具有网络组织结构特征的数据，Spagobi 提供了 Network Analysis 方式展示与分析数据。

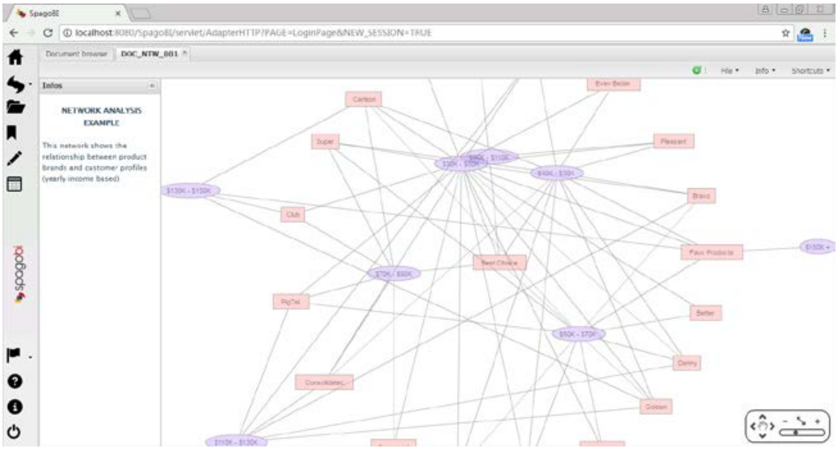


图 6.26　Spagobi 的 Networkanalysis 分析样例

由上述示例可知，经过 Spagobi 可视化后的数据十分直观、清晰，能使人更方便地进一步解读数据。实践中一般会通过如图 6.27 所示的过程，完成原始数据的最终展现。

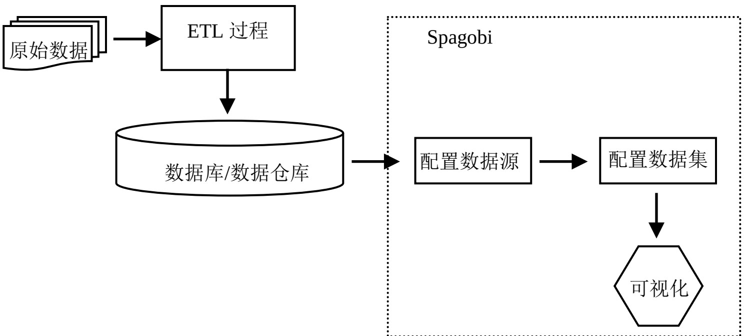
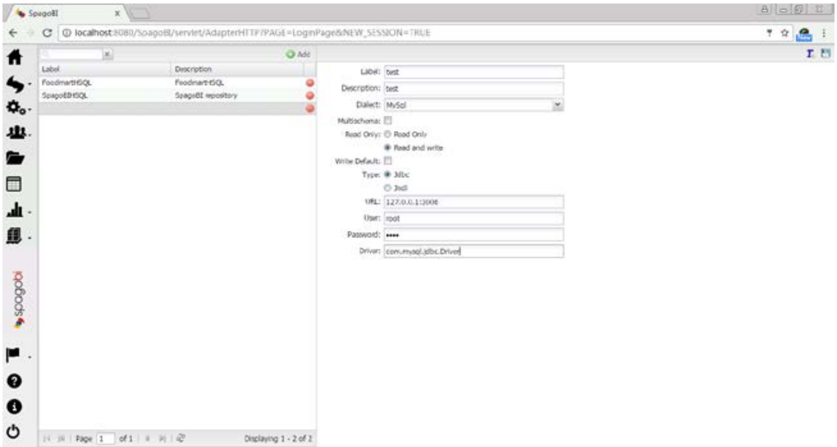


图 6.27　Spagobi 数据处理流程示意图

如图 6.27 所示，原始数据经过 ETL 过程，以一定的组织方式录入到 MySQL、HBase、Hive 等数据库或者数据仓库。Spagobi 嵌入了访问这些存储介质的驱动，在“配置数据源”的操作中，只需要指定访问地址、数据名以及密码，即可访问这些数据源。“配置数据集”指的是依据不同数据源的查询语法，获得查询结果。查询语句配置到 Spagobi 中，查询结果用作后续的可视化呈现。

如图 6.28 所示，只需输入 URL、User、Password 以及 Drive 即可完成 MySQL

数据源的配置。后续在该数据源下，建立相关数据集。



6.28　Spagobi 配置数据源

如图 6.29 所示，可以在指定数据源下建立数据集。所谓配置数据集，就是设定一些 SQL 查询语句，这些 SQL 的查询结果即是数据集。基于数据集，我们可以建立报表、网络、地图等多种形式的数据可视化模型。

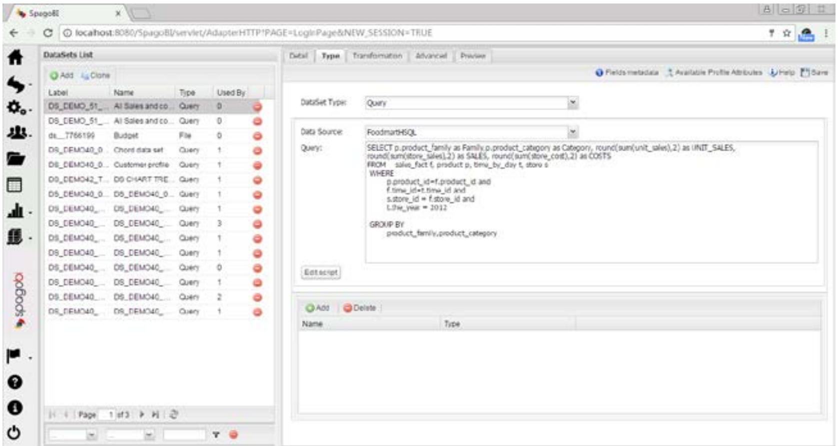


图 6.29　Spagobi 配置数据集

2．Kibana

Kibana 与 Elasticsearch、Logstash 同属于 Elastic 公司，三个组件的无缝集成，

使得 ELK 成为数据处理场景极其实用的套件。Kibana 提供的语法可以对 Elasticsearch 中的数据进行多维度检索，并且可以使用检索的数据来描绘柱状图、饼图、折线图等可视化图形。不仅如此，通过一些插件的运用，还可以完成诸如监控报警等功能。

下面我们用一个简单的例子说明 Kibana 的基本使用。

首先，在 Elasticsearch 中建立“索引”test_index，并导入若干条以下形式的数据作为测试。在 Kibana 的 Management 中配置该“索引”。

```
{
    "user": "shary",
    "date": "WedMay 31 13:50:29 CST 2017",
    "txt": "I like to use python in my work",
"python_version":"3.1.5"
}
```

如图 6.30 所示，在“Index name or pattern”输入框中输入索引名称，该项支持“*”语法代表动态索引。如果数据中存在“@timestamp”，还可以勾选“Index contains time-base devents”项，从而使数据以时间排序展现。配置完成后，在“Discover”操作栏中，可以看到该索引对应的数据如图 6.31 所示。

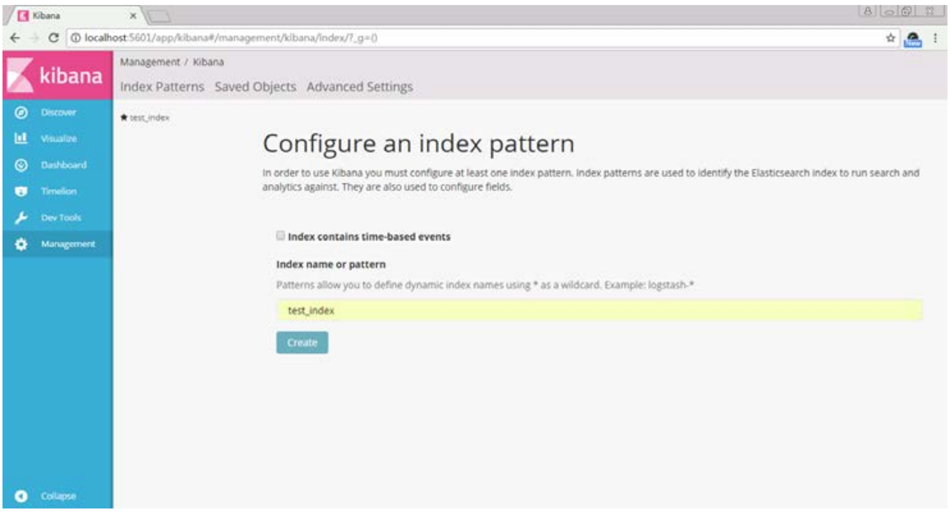


图 6.30　Kibana 选定 Elasticsearch 索引

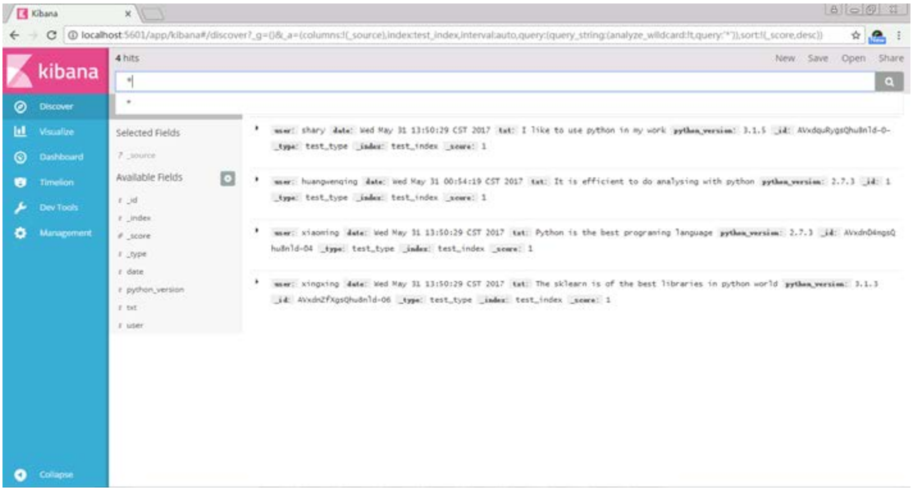


图 6.31 Kibana 中的 Discover 数据

如图 6.31 所示，在 Elasticsearch 创建的索引 test_index 中总共插入了 4 条数据。“Discover” 操作界面的上方提供了 Query 栏位，可以输入 Kibana 支持的查询语句。

如图 6.32 所示，在 Query 栏中输入 “user:huangwenqing”，即可查询到 test_index 索引中 user 对应的 Value 是“huangwenqing”的数据。更多其他的查询语句，读者可参照官网资料学习。

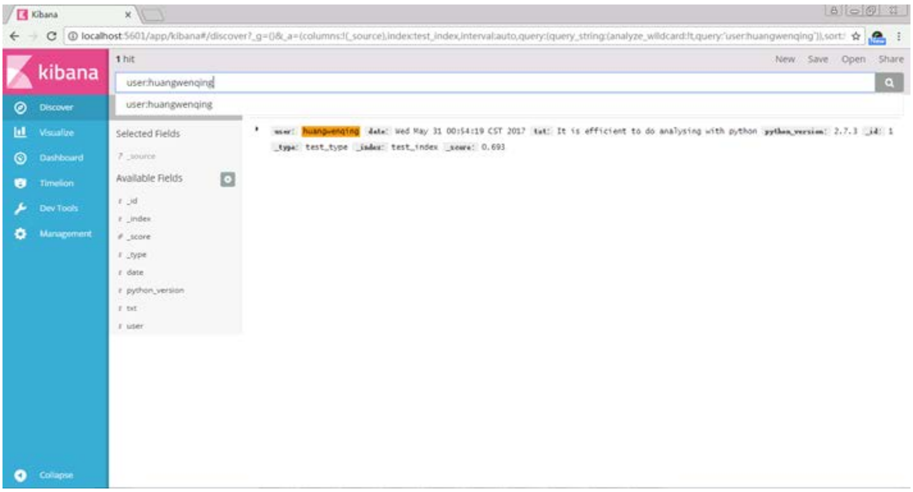


图 6.32 Query 语句查询数据示意图

“Visualize”操作选项中可以配置各种数据可视化图形，如图 6.33 所示。

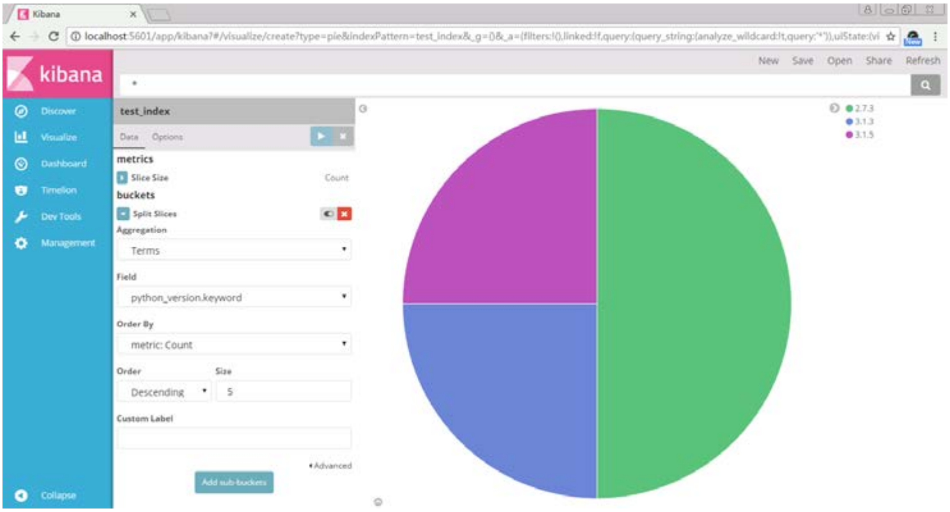


图 6.33　Python 版本分布饼图

首先，进入“Visualize”操作栏并选中“Piechart”；其次，在“Metrics”中设置“SliceSize = UniqueCount”、“Field = python_version.keyword”；在“buckets”中设置“Aggregation = Terms”、“Field = python_version.keyword”；最后，点击生成图标，即可描绘 test_index 中“python_version”字段数值分布的饼图。

Spagobi 和 Kibana 是数据处理可视化分析中较常用的两款工具。除此之外，还有 Pentaho、Openni、Bizgres 等，基本上都比较相似。诸如此类的可视化平台，在功能特点上都有所偏重，我们在做组件选型时，应当依据具体使用场景斟酌考虑。Spagobi、Pentaho 支持 OLAP 查询、而且报表样式丰富，比较适合用于商业智能；而 Kibana 与 Elasticsearch 配合，可以支持按字段查询、全文索引、实时报警，更适合用在运维方面。

6.2　大数据处理架构

大数据处理系统需要采用各种分布式组件协同构建；同时，系统不但要满足功能、性能的需求，还要兼顾伸缩性、扩展性等一系列指标，因此，如何构建大

数据处理系统对每一位工程师都是巨大的挑战。Lambda 架构、Kappa 架构以及 ELK 架构是众多架构理念中比较著名的三种架构模式，值得从事该领域的工程师学习与借鉴。

6.2.1 Lambda 架构

Lambda 架构是 Storm 创始人 Nathan Marz 在 *Big Data*: *Principle And Best Practices Of Scalable Real-Time Data Systems* 一书中提出的一种大数据的架构模式，着眼于解决大数据处理实践中存在的诸多问题，是一种架构原则和思路，而并非工程应用中的具体架构方式[34]。根据实践经验，笔者认为依据 Lambda 架构思想构建的大数据处理系统应该遵循以下准则。

（1）容错性和健壮性。系统 BUG、人为的误操作等难以避免，而在处理海量数据的系统中，这类问题很容易被成倍放大并造成巨大的损失。比如，如果集群的机器不能自动故障恢复，那么拥有 100 台机器的集群的偶尔宕机行为就会耗尽维护人员的心血；如果数据因为没有持久化而导致计算过程不可逆，那么当统计程序出现一次 BUG 时，就再也找不回原来的数据，而这个数据的量级也可能相当巨大。所以，“大数据”架构在设计时应充分全面地考虑容错性和健壮性。

（2）低延迟：系统必须保证数据的读、写和更新等操作所需要的时间在一个可控的、能满足要求的范围内——这既是硬性需求也是巨大挑战。

（3）伸缩性：在大数据的处理场景中，数据量往往会以极大的幅度增长。比如系统一个月前每天处理 2TB 的日志，一个月后就有可能每天要处理 3TB 的日志。系统的设计应该有足够的伸缩性，可以随时灵活地添加计算机以满足计算和存储要求，实现水平扩展。

（4）通用性：在大数据处理实践中，需要处理的数据来源多样，格式多样，并且还会源源不断有其他数据种类加入，因此系统必须具有一定的通用性。

（5）可扩展性：这也是其他系统的通用原则。当系统新增功能及组件时，要

使开发代价最小，实现模块与模块、组件与组件的解耦。

（6）方便查询：数据都具有一定的时效性和多维度的价值。系统要能提供方便且及时的查询，体现数据的价值。

（7）易于维护：大数据系统往往采用分布式集群，太复杂的设计往往难以维护，组件化和模块化可以减少系统的复杂程度，像 HBase + HDFS 的组合，便是模块与模块之间组合使用的典范。HDFS 提供用于数据存储的分布式文件系统；HBase 数据库搭建在 HDFS 之上，完全不用关心底层存储的实现。

（8）易于调试：数据调试的关键是要把数据流动的各个环节解耦分离；要留有原始数据，并能追溯数据的生成点。

Lambda 架构是遵循以上准则的一种通用的架构模式，它的概念图如图 6.34 所示。

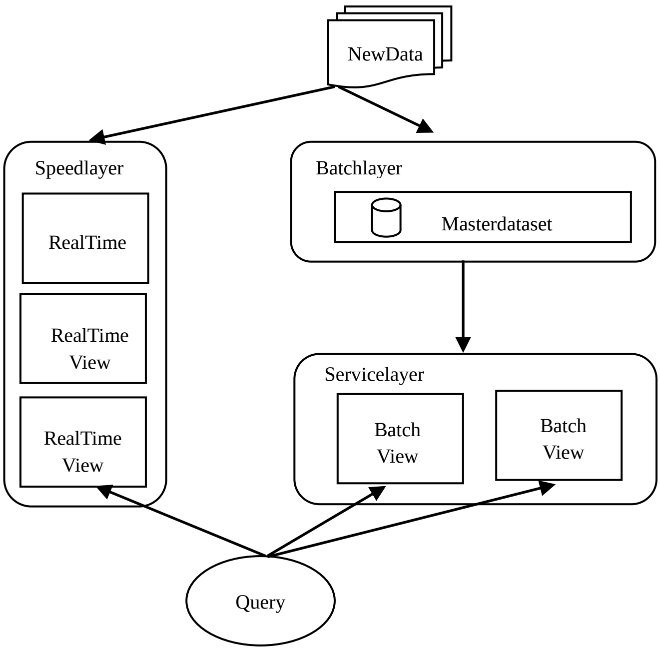


图 6.34　Lambda 架构图

图 6.34 简化地展示了 Lambda 架构的核心思想。第一，RealTime 实时处理和

Batch 离线处理相结合。时间短且小批量的数据计算从实时流增量处理，其延迟低，比如当前每秒的网站的请求数；时间长且大批量的数据计算从 Batch 离线处理，其延迟较高，比如过去一周的请求数。第二，Batchlayer 中需要存储系统的原始数据，这也是 Lambda 架构中特别强调的一点。这些数据在遇到统计程序 BUG 或其他问题时，能够重新被计算，同时，实时增量计算没有覆盖的查询部分，可以在离线的基础上补全。第三，RealTimeView 和 BathcView 要能够对外提供方便的查询。

在 Lambda 概念架构图以及各准则的指导下，笔者所参与的工程实践中，对 Lambda 架构模式做了如图 6.35 所示的组件选型和具体实现。

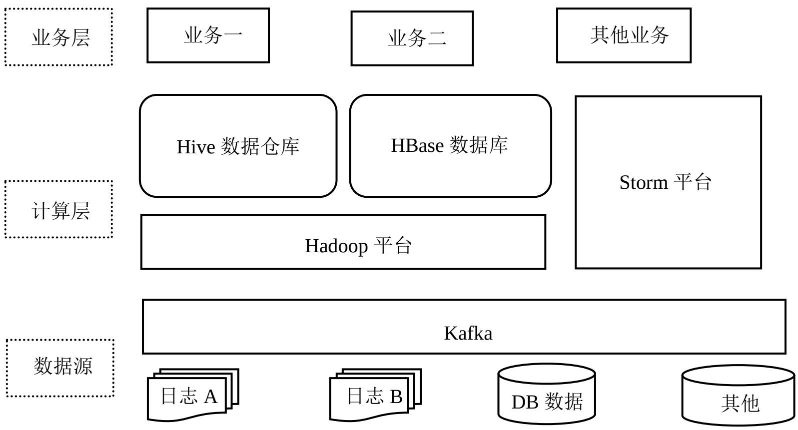


图 6.35　Lambda 架构的具体实现

数据源包含各业务的日志数据、数据库中的数据以及其他第三方数据。引入 Kafka 消息队列，实现接入数据时的缓冲和组件之间的解耦。Hadoop 和 Hive 实现数据的大批量离线处理；同时，Storm 和 HBase 实现数据的小批量实时分析。

整个数据流如下：从 Kafka 取出数据，同时传输到 HDFS 和 Storm 平台；存放在 HDFS 的数据每小时执行分布式的 ETL 过程，经该过程处理后的数据存储于 Hive 数据仓库中；传输到 Storm 平台的数据按指定维度秒级聚合后，载入 HBase 数据库，供实时查询。

架构的准则是考虑系统优劣的指标，我们以 Lambda 架构的核心思路来评判下该系统。首先用 Hadoop 和 Hive 实现历史数据的处理，Storm 和 HBase 实现了

对某些数据的实时性要求。其次，数据持久化是 Lambda 架构思想中极其重要的一点，系统中的原始数据存储在 Hadoop 的 HDFS 中正是该思想的体现。在组件选型上，Hadoop、Storm、HBase 等组件都是具有伸缩性的分布式组件，且屏蔽了分布式系统的细节，容易维护和使用。最后，Hive 的 HQL 使用类 SQL 的方式来查询数据，且 HBase 查询接口也简单易用，这些方式都极大地方便了数据的查询与使用。

图 6.35 所示的基于 Lambda 架构思想实现的具体架构方式具有一定的通用性，只需要简单变动，就可以适应于很多的大数据处理场景。比如，在需要使用 Spark 提供数据挖掘库时，可以把 Hadoop 和 Storm 替换成 Spark 和 Spark Streaming；同时，在实时流聚合后数据量很小且想要提高查询的多样性时，可以把 HBase 替换成 memSQL（一种分布式内存数据库，提供和 MySQL 一样的使用方式）。

总之，Lambda 架构从大数据处理中的各种问题出发，提出了构建大数据系统所应遵循的架构准则与思路，可以指导软件开发人员根据实际的业务场景实现更加优良的系统架构。

6.2.2　Kappa 架构

JayKreps 指出，Lambda 架构中采用“批处理方式”来弥补“实时流处理”对数据持久化和历史数据重放时的不足，是不恰当的。

首先，这种方式导致了采用该架构的工程师必须开发和维护两套数据处理体系，代价很高；其次，实时流处理是能够胜任重放和持久化数据工作的。针对这些问题，JayKreps 提出了 Kappa 架构，如图 6.36 所示。

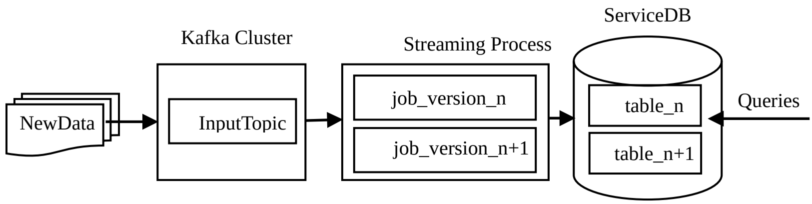


图 6.36　Kappa 架构图

由图 6.36 可知，Kappa 架构采用 Kafka 或其他方式作为数据的存储介质，实现数据的持久化。实时计算流可以由 Kafka 提供的历史数据实现任务的重新计算。摒弃批处理流程后，整个架构更加精简，也容易开发和维护。

笔者从自身的实践出发，对比 Lambda 和 Kappa 架构在实际应用中的表现，总结一些经验和思考如下。首先，Lambda 架构处理数据的规模要大于 Kappa 架构的，因为对于相同规模的历史数据，离线批处理的稳定性和计算开销都要优于实时流处理。比如 Hadoop 在处理大批量历史数据时，显然比 Storm 更加方便实用。但是，这种优势取决于集群的规模以及现有各组件自身的特点。随着技术的发展，更加完善的实时流处理取代批处理也完全有可能。其次，大数据各组件本身就具有很高的复杂度，和 Kappa 架构相比，Lambda 架构的离线和实时需要的开发和维护成本可想而知。总之，单纯对比架构的优劣并无太大意义，在实际的架构选型中，要依据实际问题，灵活运用各组件，扬长避短。

6.2.3 ELK 架构

Lambda 与 Kappa 是一种架构的理念，并给出对组件选型的一些参考。而 ELK 架构是一种具体的架构方式，常用于大数据后台系统监控、日志分析等运维领域。

其中“E”即前面介绍的 Elasticsearch，负责数据的存储、分析；“L”即前面介绍的 Logstash，负责从各数据源搜集、过滤并传输数据；“K”代表 Kibana 组件，是基于 Web 的图形界面，调用 Elasticsearch 提供的 Restful 接口提供可视化交互，允许用户创建报表、查询数据。

ELK 三种组件协同工作，实现了搜集、传输、过滤、存储、统计及展现的各个环节，其常用的一种架构模型如图 6.37 所示。

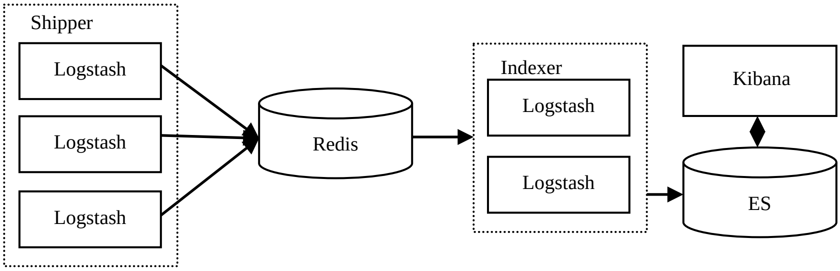


图 6.37　Logstash 的应用部署

图 6.37 中，Shipper 模块的各个 Logstash 负责接收、过滤和传输数据。Redis 通过发布-订阅或者 List 列表的方式，向 Indexer 提供数据，起到了缓存和解耦的作用。我们也可以用 Kafka 替换 Redis，这样当后端集群出现故障时，数据可以在其中存放一定的时间，确保数据的可靠性。Indexer 负责把数据传输到 Elasticsearch，由其存储并对数据建立索引。Kibana 通过 Restful 接口与 Elasticsearch 交互，实现数据的可视化操作。

ELK 架构适合处理数据量不大的准实时场景。首先，相比于 HBase，Elasticsearch 的查询模型更丰富，但是，为了实现全文索引及一些统计功能，它的数据冗余度较高，数据的实际存储空间可能远大于数据本身的大小；同时，Elasticsearch 的读写速度也略逊一筹（当然，这是指在一般的使用场景中；在实际应用中，集群的规模、索引的建立都会影响两者的读写性能），这使得它更适合于一些数据量不大（和 HBase 相比）、需要丰富的查询模式的准实时场景。在笔者的实践中，ELK 架构常用于搜集和分析后台日志，实现系统的实时监控和其他运维工作。

6.3　项目设计

（1）背景说明

随着 3D 电影的推出，越来越多的观众愿意走进电影院，享受视觉的饕餮盛宴。然而，由于信息的不对称，导致线下售票和购票的过程存在诸多问题。针对这些痛点，某 IT 公司想研发电影票网络自助售卖平台，在影院和观影用户之间搭

建桥梁。网络售票的方式不但便利了观众，也极大降低了电影院的人力成本。

（2）需求说明

基本功能需求如下。

① 电影院和观众需要申请、注册成为平台用户。

② 电影院可以发布电影场次、时长、坐席信息。

③ 电影院可以发布零食、饮料小商品信息。

④ 用户可以在线选场次、坐席，支付电影票费用。

⑤ 用户可以购买电影院售卖的零食、饮料等商品。

⑥ 用户可以在电影开始前 30 分钟退票，30 分钟内无法退票。

⑦ 用户可以在购票区留言、点赞以及写影评。

数据统计需求如下。

① 离线统计每天各场次电影的上座率、票房。

② 离线统计每天各种零食、饮料等小商品的售卖数量。

③ 实时统计每部电影的好评率。

④ 实时统计当前场次的剩余席位。

⑤ 实时统计当前各零食套餐的剩余份数。

⑥ 实时统计当前用户已购买过的电影票数量和座次分布。

⑦ 网站首页需针对用户实施电影和影院的个性化推荐。

⑧ 预测电影的票房，以合理安排场次。

性能需求如下。

① 网页常规操作响应时长在 1 秒以内。

② 实时数据统计的延迟在 1 秒以内。

③ 离线数据统计的延迟是 1 天。

（3）量级评估

此处出于展示大数据架构的考虑，我们假设有如下量级。

- 首页 PV 的峰值 100 万 QPS。
- 每小时 30 万次订票。
- 每秒产生 500 万条日志。
- 每天产生 10TB 的日志量。

（4）架构考量

① 功能：实现用户所有功能及统计需求。

② 性能：达到用户要求的性能指标。

③ 伸缩性：考虑到业务的增长，应该可以随时添加机器完成计算与存储能力的扩展。

④ 扩展性：架构各组件之间可以解耦，能灵活地增加和减少各功能模块。

⑤ 安全性：数据的备份、数据的加密以及完备的权限管理措施。

⑥ 稳定性：对系统进行有效的监控，能及时故障报警等。

（5）设计思路

架构目标是达成上述所示的考量指标。架构的过程涉及系统的方方面面，在《软件架构设计》一书中，作者以五视图法来拆分架构，即“逻辑架构”“运行架构”“开发架构”“数据架构”和“物理架构”[35]。在该书中，作者强调“风险驱动”，即架构是用来提前规避可能导致项目失败的风险点的；以风险驱动设计，避免冗余，做到恰如其分[36]。笔者认为，架构是条理清晰且有迹可循的设计，毕竟某些“共性”在每个架构中都会存在，比如“分层”“组件化”“模块化”和“负载均衡”等；同时，架构也掺杂了很多主观臆断，比如对“业务的发展速度”的判断决定了是采用“重”架构还是“轻”架构，“团队的技术领域”和“技术生态圈的成熟度”决定了组件的选型等。因此，没有永恒完美的架构，只有随着业务发展而不断演化完善的架构。

首先，针对该电影票售卖平台的设计，笔者查阅相关架构资料作为参考，拟定了负载均衡、分层、微服务、组件等基本框架；其次，依据业务量和其他因素，完成组件选型；最后，详细设计各重点模块的实现流程，并优化架构各个细节。

当然，这三步是一个不断循环斟酌的过程，并且掺杂了各种非技术因素的考虑，最终得出的架构逻辑如图 6.38 所示。

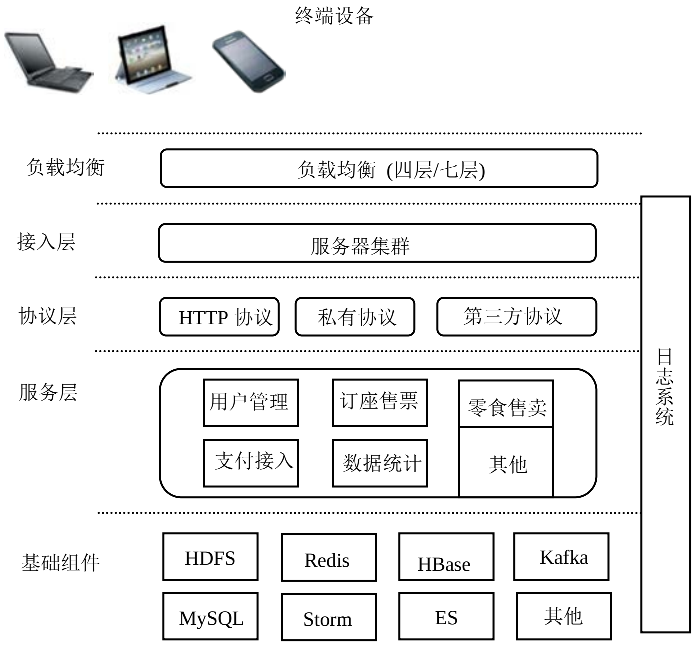


图 6.38　电影票在线销售平台架构图

负载均衡，让流量平摊到多台服务器上，解决了高峰值请求时单点故障的问题，常采用“四层”与“七层”配合的组合负载策略。“四层”指的是根据 IP+端口的方式实现均衡策略；“七层”指的是根据 URL 等应用层信息实现均衡策略。前者处理请求的量级大，但是负载策略单一，常用的组件包括 LVS、F5 等；后者处理请求的量级相对较小，但是可以采用 URL、参数等来实现多样的负载策略，常用的组件包括 Nginx、HAproxy 等。通常情况下，四层负载均衡接受请求，分发到七层负载均衡的服务器上，由七层负载均衡根据一定策略转发请求。

接入层，是一个服务器的集群，经过负载均衡的请求会被路由到某台服务器。这种设计让服务器的承载能力实现横向扩展。

协议层，负责多种协议的适配工作。比如浏览器会以 HTTP 的方式发送请求、

物联网硬件设备也可能以 Coap 协议来发送请求等。当前各系统大多是基于 HTTP 协议，以提供 Restful 接口的形式对外开放自己的服务。即使如此，把协议层独立出来，配合下层微服务的设计，可以使整个架构更加清晰。

服务层，采用微服务的思想，从业务逻辑的角度把系统划分成多个独立模块，每个模块以 Restful 接口独立对外提供服务。如何划分则取决于业务本身，独立的业务划分成独立的模块。微服务的设计让这个系统充分解耦，使系统具有更好的扩展性和灵活性；同时，也解决了代码糅杂在一个模块中在工程实现上所带来的诸多问题——比如少量代码更新也会导致较长时间的编译、或者某块代码 Bug 导致整个服务的崩溃等。

基础组件，负责搭建与维护多种组件，并对外提供稳定的服务，比如 Storm、Kafka 等。这样对于写业务逻辑的工程师而言，不需要处理 NoSQL、Storm 等分布式平台的搭建、维护以及调优等工作，完全把这些组件当成黑盒来使用；同时，负责基础组件的工程师，也可以只专注于这些通用组件的学习与研究。

日志挖掘与监控系统，负责数据应用和运维监控。数据应用包括用户数统计、用户观影统计、个性化推荐等；运维监控包括请求异常报警、系统状态报表等。该内容是本节围绕大数据处理需要重点讲述的内容，详细设计如图 6.39 所示。

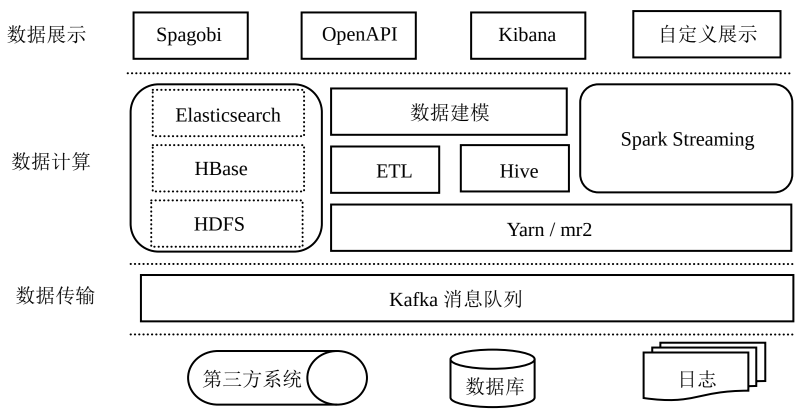


图 6.39　日志系统架构图

图 6.39 简要描述了整个日志系统的层次结构，模块功能和组件选型。“请求

日志”“用户信息”“第三方数据”等不同数据源，经由 Rsync 或 Kafka 传输，在计算模块进行数据处理,并以多种方式对外输出。详细的数据流图如图 6.40 所示。

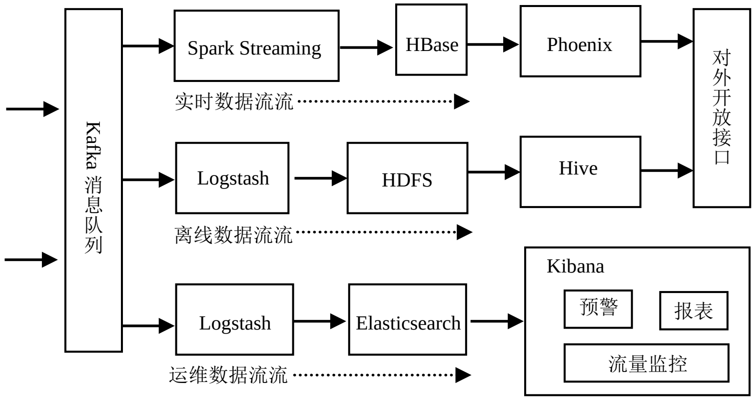


图 6.40 日志系统数据流图

图 6.40 描述了数据在该日志系统的数据处理流。数据传入 Kafka 消息队列，依据业务需求分为“实时数据流”“离线数据流”“运维数据流”。

首先，对该日志系统而言，Kafka 消息队列接受多种数据源，起到了缓冲和解耦的目的。“缓冲”的作用显而易见，大流量的日志数据可以暂存到 Kafka 中，后端系统根据计算速度，按需拉取数据，解决了高峰值导致的系统崩溃。“解耦”使异构数据源和后端系统遵循 Kafka 的协议，输入和输出数据，避免了多种接入方式带来的系统复杂性。同时，读者也应该意识到，Kafka 的引入必然一定程度上会降低实时性，这在日志系统中可以忽略，但是在其他的场景中应灵活运用。比如笔者参与的某流量拦截的项目中,在请求高峰时,即使只有 1 秒左右的延迟，也会透过巨大的流量，导致后端系统过载；此时应考虑使用负载均衡的方式横向扩展达到更实时的处理，或者选用例如 ZeroMQ 的内存消息队列，会更加稳妥。

“实时数据流”中引入 Spark Streaming 的目的是为了对数据做实时的初步聚合。比如针对 1 秒钟有 10,000 条日志数据的情况，按时间戳、影院、场次的维度聚合之后可能只有 300 条。选择 HBase 作为 NoSQL 数据库的原因有以下几点。

① 与架构中其他组件 Hive、MR 等同属于 Hadoop 生态圈，具有大量的用户

和很多成功的应用案例。因此，HBase 必然具有丰富的文档、活跃的论坛、针对无数“坑”的解决方案。

② 如 6.1.2 节所介绍，HBase 具有大容量、可扩展、高吞吐以及非结构存储的功能特点，非常符合海量日志数据的处理场景。

③ 与 Elasticsearch、Redis、MongoDB 相比，HBase 在性能、处理数据量以及稳定性上都具有明显的优势。

当然，HBase 的缺点也十分明显，就是查询模式十分单一，不支持 SQL 化语义查询。以笔者的经验来说，在绝大多数的数据处理场景中，对 SQL 的支持都是不可或缺的。因此，在不想舍弃 HBase，又希望得到丰富查询的矛盾中，当前常用的解决方案是引入 Phoenix，或者与 Phoenix 类似的组件，用意在于依托 HBase 实现一套支持 SQL 语义的中间件。

“离线数据流”中，Logstash 从 Kafka 拉取数据，并保存到 HDFS 中；经由 ETL 过程，最终载入 Hive 数据仓库。Logstash 的其他替代选型还有 Flume 等一些常用组件。由于在“运维数据流”中，ELK 是最常用的组合套件，因此，这里的数据通路也选择了 Logstash，以减少维护的代价。HDFS 作为分布式文件系统，毋庸置疑，没有其他更优的组件可以替代它。Hive 的定位是“数据仓库”，用于处理更大和更详细的历史数据。需要注意的是 HBase 的定位是“数据库”，写入和查询的速度远高于 Hive，而量级却无法与之相比（关于“数据仓库”和“数据库”的定位，读者可以细细考究）。Lambda 架构的思想是引入离线流用以存储原始数据、实施数据回放，也作为大数据量无法实时处理的一种折中方案。同时，这些离线数据更大的作用是可以作为训练数据集,实施数据建模,挖掘数据价值。

“运维数据流”中的 Logstash、Elasticsearch 以及 Kibana 是自动化运维的常用套件。Logstash 用于日志的传输，Elasticsearch 用于数据的存储于查询，Kibana 用于数据的可视化。数据运维的场景往往具有以下特点：① 数据量级较小；② 对详细数据的各字段查询；③ 准实时要求；④ 多样的查询模式。相比于 HBase 和 Phoenix 组合，Elasticsearch 在吞吐量以及处理数据量级上都略逊一筹；但是，它能以准实时的方式，支持全文检索和多种查询方式，维护的成本也较低；同时，

能与 Logstash、Kibana 组件无缝集成。因此，ELK 成为运维场景中的经典组合。

“数据建模”一般基于离线历史数据来训练有效的模型。历史数据可能是来自一小时，一天或者更长的时间内的数据。利用分布式平台，实现对海量数据挖掘建模通常有以下两种思路：第一，如果数据之间没有关联，可先拆分，之后在各台机器上单独建模。比如先获取各家电影院每天的售票数量，然后运用 Sklearn 等机器学习库实施建模；第二，如果数据之间是一个整体，可基于 MapReduce 模型来实现分布式算法。比如“朴素贝叶斯”中各统计值可以基于 MR 来计算，不过，为了避免重复“造轮子”，通常会选用一些分布式的库或者平台，比如 TensorFlow、Mahout 等。数据建模方式如图 6.41 所示。

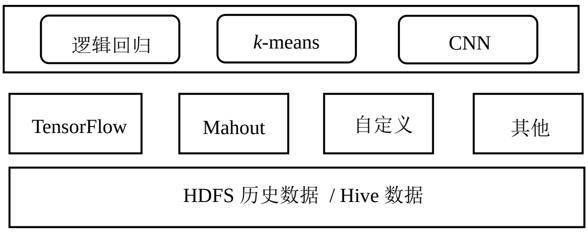


图 6.41　数据建模方式示意图

参考文献

[1] 王燕.《应用时间序列分析》.[M].中国人民大学出版社, 2015.12

[2] Pang-NingTan, Michael Steinbach, VipinKumar. *Introduction to Data Mining*. [M]. 机械工业出版社,2010.09.01

[3] https://zh.wikipedia.org/wiki/ETL.[EB/OL]

[4] https://www.mathworks.com/.[EB/OL]

[5] https://www.ibm.com/analytics/us/en/technology/spss/.[EB/OL]

[6] http://www.stata.com/company/.[EB/OL]

[7] https://www.sas.com/zh_cn/home.html.[EB/OL]

[8] http://www.eviews.com/general/about_us.html.[EB/OL]

[9] https://zh.wikipedia.org/wiki/Python.[EB/OL]

[10] https://www.ibm.com/blogs/watson-health/the-5-vs-of-big-data/.[EB/OL]

[11] ViktorMayer-Schonberger, KennethCukier.*BigData: ARevolutionThatWillTransformHowWeLive, Work, andThink*.[M].EamonDolan, 2014.03

[12] http://www.scipy.org/.[EB/OL]

[13] 盛骤，谢式千，潘承毅.《概率论与数理统计》.[M]. 高等教育出版社, 2008.06

[14] https://www.kaggle.com/.[EB/OL]

[15] https://medium.com/towards-data-science.[EB/OL]

[16] Rosenblatt, Frank.*The Perceptron: A Probabilistic Modelfor Information Storageand Organizationin the Brain, Cornell Aeronautical Laboratory*. [C]. Psychological Review, 1956

[17] WarrenS. McCulloch, Walter Pitts. *A logical calculus of the ideas immanent in nervous activity*. [C].1943

[18] G. E. Hinton, Simon Osindero, Yee-Whye Teh. *A fast learning algorithm for deep belief nets*. [J].Neural Computation,2006.

[19] Donald Olding Hebb. *The Organization of Behavior*.[M].1949

[20] Alan Mathison Turing. *Computing Machinery and Intelligence*. [J].1950

[21] Frank Rosenblatt. *Principles of Neuro dynamics: Perceptrons and the theory of brain mechanisms*. [M]. 1961

[22] Marvin Minsky,Seymour Papert Perceptrons. *an introduction to computation algeometry*.[M].1969

[23] Paul Werbos. *New Tools for Prediction and Analysis in the Behavioral Sciences*. [D].1976

[24] Geoffrey Everest Hinton,David E. Rumelhart,Ronald J. Williams. *Learning representations by back-propagating errors*.[J].1986

[25] Yann LeCun. *Backpropagation Applied to Handwritten Zip Code Recognition*. [J].1989

[26] Corinna Cortes,Vladimir Vapnik. *Support-Vector Networks*.[J].1995

[27] Geoffrey Everest Hinton. *Contrastive Divergence Training Products of Experts by Minimizing CD*. [J].2002

[28] Geoffrey Everest Hinton,Simon Osindero,Yee-Whye Teh. *A fast learning*

algorithm for deep belief nets.[J].2006

[29] http://yann.lecun.com/exdb/mnist/.[DB]

[30] https://www.tensorflow.org/.[EB/OL]

[31] http://www.cs.toronto.edu/~kriz/cifar.html.[DB]

[32] Michael McCandless, Erik Hatcher. *Lucene in action*.[M].2011.06

[33] 董西成.《Hadoop 技术内幕：深入解析 MapReduce 架构设计与实现原理》.[M].2013.05

[34] Nathan Marz,James Warren *BigData-Principle And Best Practices Of Scalable Real-Time Data Systems*. [M].2015.05

[35] 温昱.《软件架构设计》.[M].2007

[36] George Fairbanks, *Just Enough Software Architecture*. [M].2013.09